Théorie des graphes

L3 TECHNOLOGIES DE L'INFORMATION

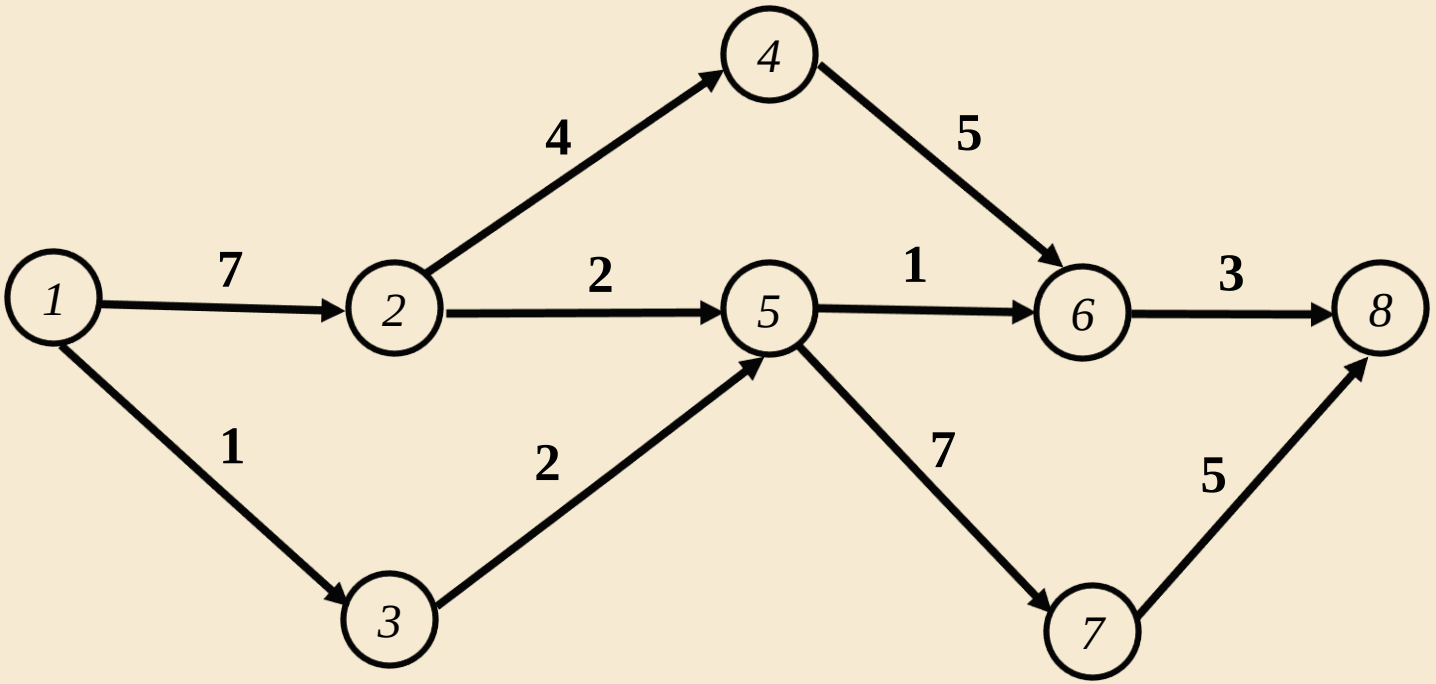
DR ABDALLAH EL KHYARI
ABDALLAH.ELKHYARI@GMAIL.COM

INTRODUCTION

Analyse des problèmes (1)

Recherche de chemin

- **Un ou des chemin(s) reliant les sommets 1 à 8**



Analyse des problèmes (2)

Problème ExisteChemin :

- Instance : un graphe orienté et valué G, et 2 sommets 1 et 8.
- Question : existe-t-il un chemin dans G qui relie le sommet 1 à 8 ?

Problème TrouveChemin :

- Instance : un graphe orienté et valué G, et 2 sommets 1 et 8.
- Question : trouver un chemin dans G reliant le sommet 1 au sommet 8.

Problème TousLesChemins :

- Instance : un graphe orienté et valué G, et 2 sommets 1 et 8.
- Question : trouver tous les chemins dans G qui relient le sommet 1 à 8.

Problème PlusCourtChemin :

- Instance : un graphe orienté et valué G, et 2 sommets 1 et 8.
- Question : trouver un plus court chemin dans G reliant le sommet 1 à 8.

Analyse des problèmes (3)

Problème de décision

- C'est un problème qui prend en entrée une instance, et répond en sortie "oui" ou "non" suivant si l'instance répond au problème ou pas.

Problème d'existence (recherche d'une solution)

- C'est un problème qui comporte une question ou plutôt une injonction de la forme *« trouver un élément tel que …»* dont la réponse consiste à fournir un tel élément.

Problème d'énumération (recherche de toutes les solutions)

- C'est un problème qui comporte une question ou plutôt une injonction de la forme *« trouver tous les éléments tel que …»* dont la réponse consiste à fournir un ensemble d'éléments.

Problème d'optimisation (recherche de la meilleure solution)

- C'est un problème d'existence qui consiste à fournir une solution qui optimise un certain critère (coût, profit, …).

Problème décision/optimisation

Problèmes de décision

- Réponse = oui ou non
- Exemple : Peut-on colorier ce graphe avec 4 couleurs ?

Problèmes d'optimisation

- Réponse = valeur qui optimise une fonction objectif donnée
- Exemple : Plus petit nombre de couleurs permettant de colorier ce graphe ?

Problème de décision vs problème d'optimisation

- Existe t'il une meilleure solution ?
- Exemple : Peut-on colorier ce graphe avec moins de 5 couleurs ?

Remarque

- L'équivalence entre ces deux problèmes suppose que la démonstration d'existence repose sur un argument constructif

Problèmes & algorithmes

Définitions

- Un **problème de décision** est dit **décidable** s'il existe un algorithme, une procédure mécanique qui termine en un nombre fini d'étapes, qui le décide, c'est-à-dire qui répond par oui ou par non à la question posée par le problème
- S'il n'existe pas de tels algorithmes, le problème est dit **indécidable**

Remarques

- Un problème **décidable** peut admettre plusieurs algorithmes qui le résolvent
- Exemple : recherche d'un élément dans une liste

Complexité algorithmique (1)

Définition

- La complexité d'un algorithme A est une fonction $C_A(N)$, donnant le nombre d'instructions caractéristiques exécutées par A dans le pire des cas, pour une données de taille N.

Pire cas

- pour N fixé, on considère la donnée donnant le plus de travail à l'algorithme. Une complexité moyenne serait utile mais difficile à calculer.
- La complexité en moyenne nécessite une connaissance de la distribution probabiliste des données.

Complexité algorithmique (2)

Algorithme de tri par sélection

- Le pseudo-code de l'algorithme s'écrit :

```
procédure tri_selection(tableau t, entier n)
    pour i de 1 à n - 1
        min = i
        pour j de i + 1 à n
            si t[j] < t[min] alors min = j
        fin pour
        si min ≠ i alors échanger t[i] et t[min]
    fin pour
fin procédure
```

- Complexité de l'algorithme : **$O(n^2)$**

Complexité algorithmique (3)

Trier un tableau de n éléments

- Algorithme de tri par sélection : **$O(n^2)$**
- Algorithme de tri par insertion :
  - dans le meilleur des cas (tableau déjà trié) : **$O(n)$**
  - dans le pire des cas (tableau trié à l'envers) : **$O(n^2)$**
- Algorithme de tri rapide (quicksort) :
  - dans le meilleur des cas (pivot = médiane) : **$O(n.\log(n))$**
  - dans le pire des cas (pivot = elt max ou min) : **$O(n^2)$**
- Algorithme de tri par tas : **$O(n.\log(n))$**

Classification des problèmes (1)

La classe P (Deterministic Polynomial time)

- L'ensemble des **problèmes de décision** qui peuvent être résolus par un algorithme polynômial

La classe NP (NonDeterministic Polynomial time)

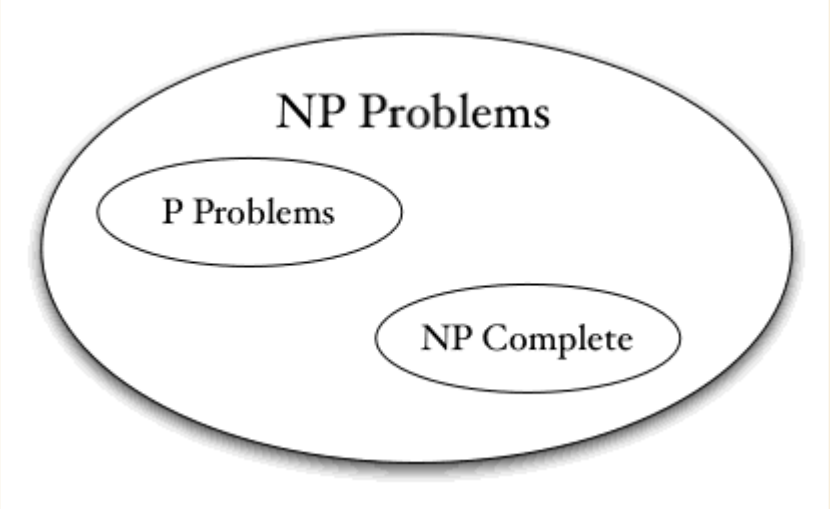
- L'ensemble des **problèmes de décision** pour lesquels on possède pour chaque « oui » (instance I) un schéma de vérification, vérifiant en temps polynômial la validité de la réponse « oui »

Problème NP-Complet

- Un **problème de décision** A dans NP est NP-complet si tous les autres problèmes de la classe NP se transforment polynomialement dans le problème A.

Classification des problèmes (2)

Relations entre P et NP



- P $\subseteq$ NP
- Conjecture : P $\neq$ NP
- 1 million de dollars à gagner !
- http://www.claymath.org/millenium-problems/

RECHERCHE OPERATIONNELLE

Définitions (1)

Cambridge Dictionary

- Operational research UK (US operations research): The systematic study of how best to solve problems in business and industry

Wikipedia

- Operations research, operational research, or simply OR, is the use of mathematical models, statistics and algorithms to aid in decision-making

Roadef (Société Française de RO-AD)

- Recherche Opérationnelle : approche scientifique pour la résolution de problèmes de gestion de systèmes complexes

Définitions (2)

Plus précisément

- Méthodes scientifiques pour résoudre des problèmes d'optimisation liés aux organisations du monde réel
- Une discipline à la croisée des mathématiques et de l'informatique :
  - Prolongement de l'algorithmique
  - Manipulation des structures plus élaborées : graphes, polyèdres, …
  - Domaine d'application de la théorie de la complexité algorithmique
- Une boîte à outils de méthodes, tant positives que négatives, pour aborder sainement et sereinement les problèmes d'optimisation

Secteurs d'application

Applications

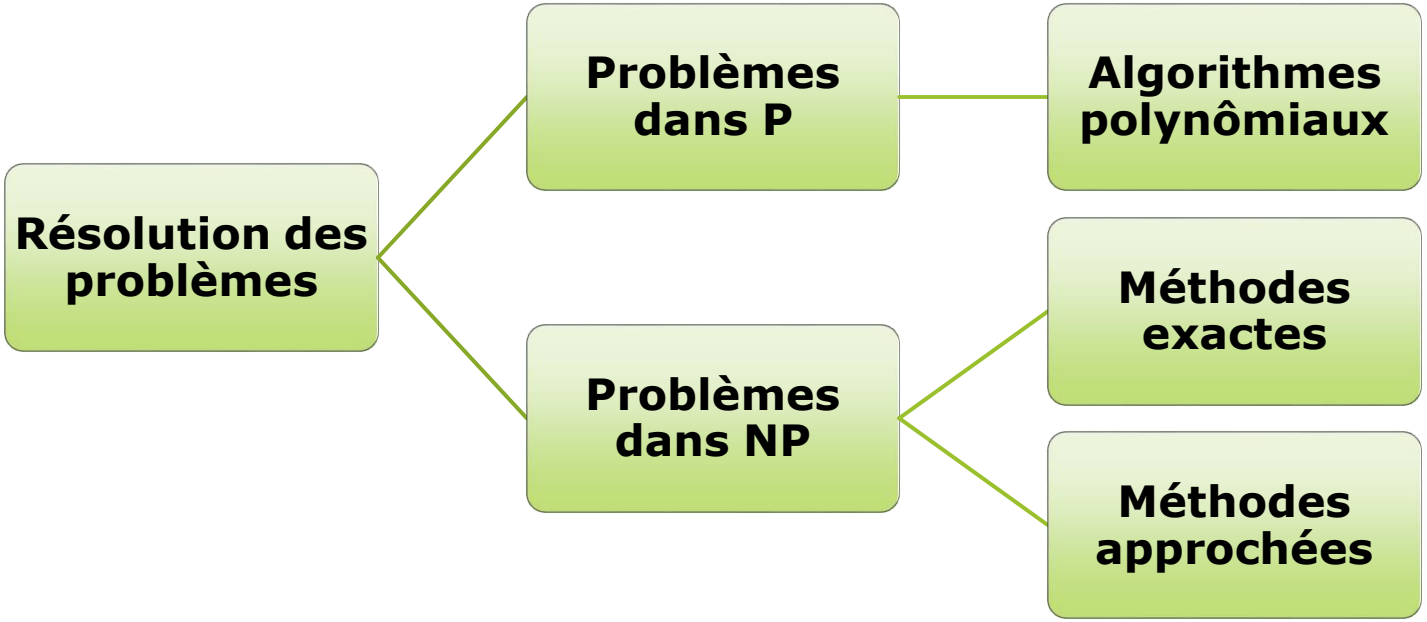
- Industrie minière
- Transports : routier, ferroviaire, fluvial, aérien
- Gestion de la chaîne logistique
- Gestion de l'énergie
- Santé
- Planification de la production
- Problèmes d'ordonnancement
- Télécommunications
- Economie et finances
- Emplois du temps
- …

Entreprises & RO

Grands groupes français avec des pôles R&D en RO

- Airfrance
- SNCF
- EDF
- France Telecom
- Bouygues
- GDF Suez
- La Poste
- Renault
- Air Liquide
- SFR
- Google

Résolution d'un problème



Face à un problème

Comprendre le problème posé :

- Contraintes, objectifs, simplifications

Modéliser le problème :

- Graphes, programmation linéaire, branch & bound (algorithme par séparation et évaluation), heuristiques, …

Connaitre les propriétés du modèle :

- Étude de complexité : que peut-on espérer pour le temps de résolution imparti ?

Résoudre le problème :

- Utiliser un solveur
- Mise au point d'algorithmes

Interpréter la solution :

- Interpréter la solution numérique obtenue en termes concrets

Quelques problèmes

Problèmes polynômiaux

- Programmation linéaire en continues
- Trouver le plus court chemin entre 2 sommets
- Trouver un chemin passant 1 fois par chaque arc (Eulérien)
- Maximiser un flot de véhicules dans un réseau de transport

Problèmes NP-complets

- Programmation linéaire en nombre entiers
- Problème du sac à dos
- Problèmes d'ordonnancement
- Problème du voyageur de commerce
- Trouver un chemin passant 1 fois par chaque sommet (Hamiltonien)
- Problème de la clique : Trouver k sommets connectés 2 à 2 par des arêtes
- Colorier les sommets d'un graphe

Contenu du cours

Contenu

- Graphes orientés
  - Définitions
  - Quelques exemples de graphes
- Graphes non-orientés
  - Définitions
  - Quelques exemples de graphes
- Fermeture transitive d'un graphe
- Graphe sans circuit
- Noyau d'un graphe
- Recherche d'un chemin de longueur min ou max
- Algorithmes de résolution :
  - Ford, Bellman-Kalaba, Djikstra
- Flot de coût minimum
- Problèmes d'ordonnancement

Organisation du cours

Volume horaire

- CM : 22 heures
- TD : 20 heures
- TP : mini-projet

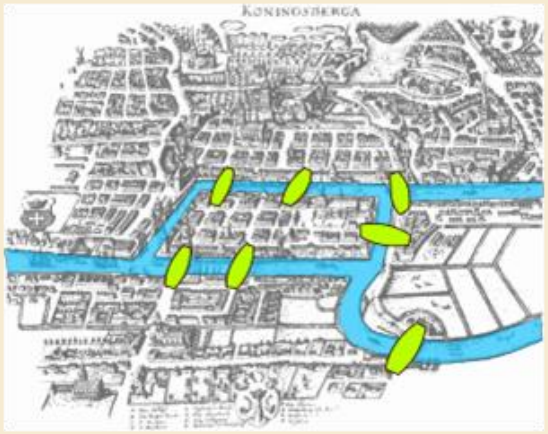
- Contrôles continus

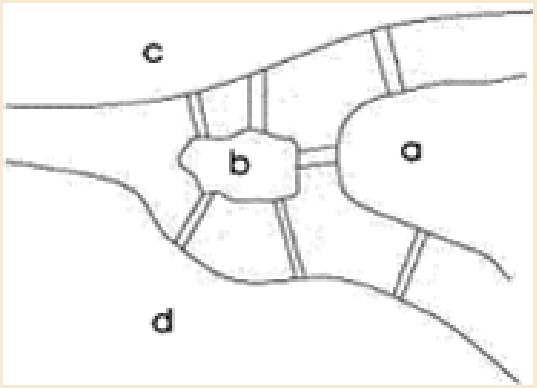
THEORIE DES GRAPHES

Origine des graphes

Le problème des sept ponts 1/3

- La ville de Königsberg est construite autour de deux îles
- Ces iles sont reliées entre elles par un pont.
- Six autres ponts relient les rives de la rivière à l'une ou l'autre des deux îles



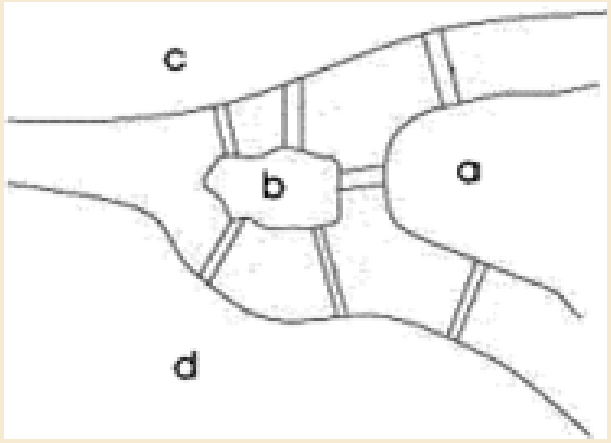


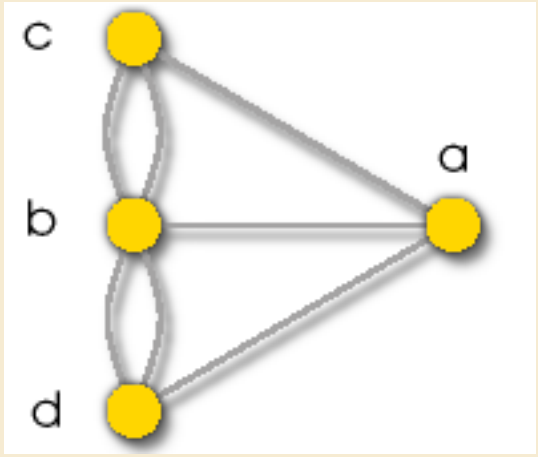
- Le problème était le suivant : Peut on se promener en passant une fois et une seule fois par tous les ponts ?

Origine des graphes

Le problème des sept ponts 2/3

- Le problème des ponts de Königsberg se modélise par un graphe



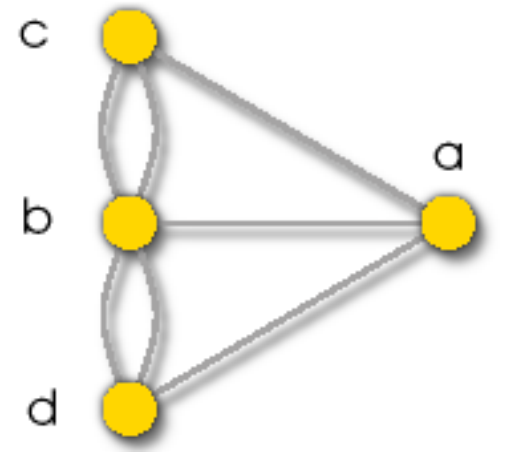


- Le problème revient à chercher un cycle eulérien : chaîne passant par toutes les arêtes du graphe une et une seule fois, et revenant à son point de départ.

Origine des graphes

Le problème des sept ponts 3/3

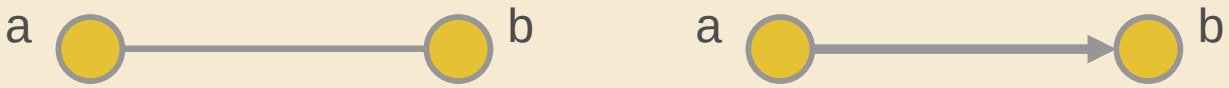
- Léonhard Euler a prouvé que le problème est insoluble
- Léonhard Euler (1736) a démontré que pour qu'un graphe admet un cycle Eulérien, il faut que le graphe soit connexe et ne possède aucun sommet de degré impair



Introduction

Introduction

- Les graphes capturent principalement la notion de relations binaires



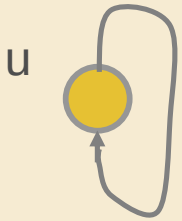
- R(u,v) est vraie
- Une relation peut dans la vie courante lier des objets divers :
  - Ali connaît Adam ---- graphe non orienté
  - Samira est plus grande que Amina ---- graphe orienté
  - E1 est client de E2 ---- graphe orienté
  - Il y a une route entre telle et telle ville ---- graphe non orienté

Graphes orientés

Définition

- Un graphe orienté G est composé :
  - V ensemble de sommets
  - A ensemble des arcs
- $A \subseteq \{(u,v): u,v \in V\}$
- $(u,v) \neq (v,u)$: relations non symétriques

Un arc de u vers v

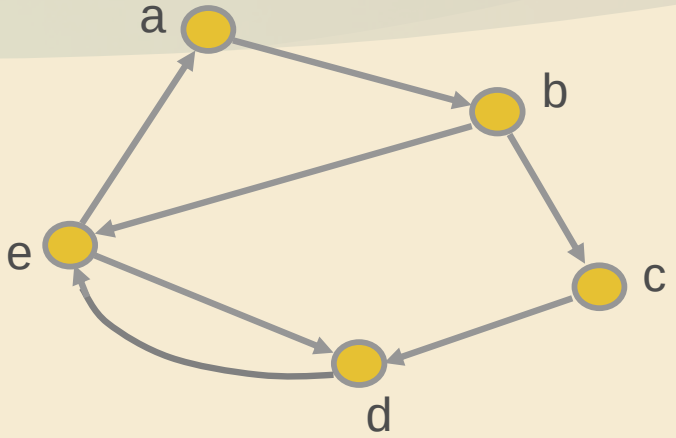


Une boucle

Graphes orientés

Exemple

- V=(a,b,c,d,e)
- A={(a,b),(b,c),(b,e),
- (c,d),(d,e),(e,d),(e,a)}



- **Remarque** : Un arc non orienté peut toujours être transformé en une situation où l'on n'a que des arcs orientés.

Exercice 1

- Construire un graphe orienté dont les sommets sont les entiers compris entre 1 et 12 et dont les arcs représentent la relation « être diviseur de »

Graphes orientés

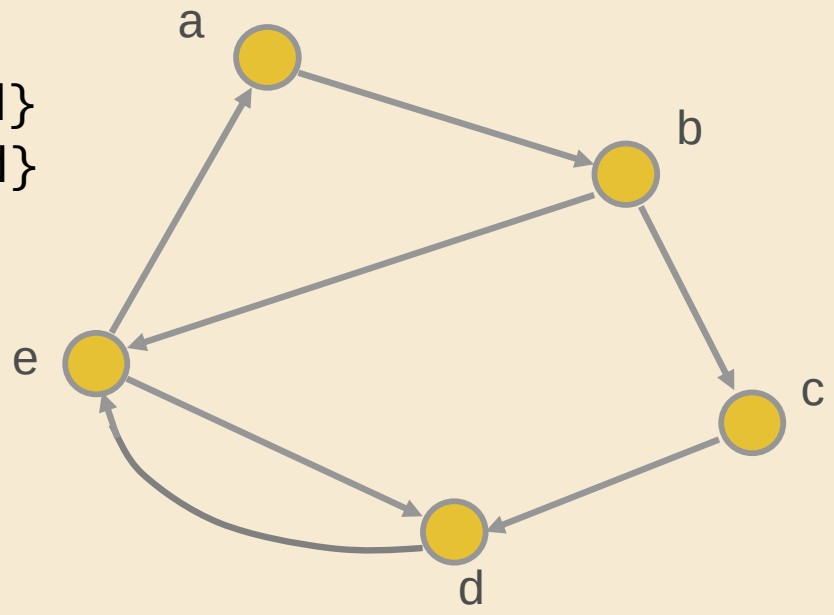
Définitions

- Soit u un sommet de G=(V, A)
  - $N^+(u)=\{v: (u,v) \in A\}$: voisins sortants de u
  - $N^-(u)=\{v: (v,u) \in A\}$: voisins entrants de u
  - Le demi-degré extérieur d'un sommet est le nombre d'arcs adjacents qui en partent. On le note $d^+(u)$
    - $d^+(u) = |N^+(u)|$: v est un successeur de u
  - Le demi-degré intérieur d'un sommet est le nombre d'arcs adjacents qui y arrivent. On le note $d^-(u)$
    - $d^-(u) = |N^-(u)|$: v est un prédécesseur de u
  - Le degré d'un sommet est le nombre d'arcs qui lui sont adjacents. On le note d(u)
    - $d(u) = d^+(u) + d^-(u)$.

Graphes orientés

Exemple

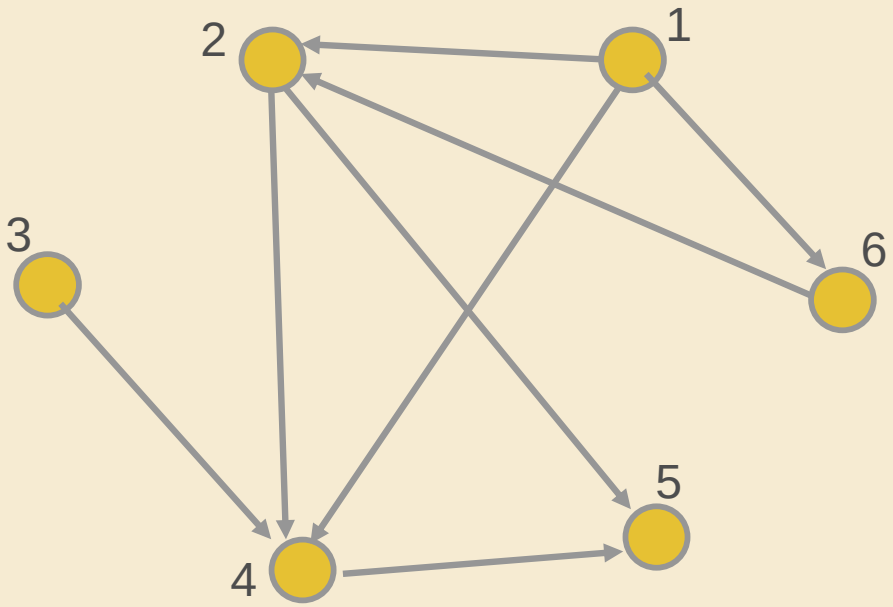
- $N^+(e)=\{v: (e,v) \in A\} =\{a,d\}$
- $N^-(e)=\{v: (v,e) \in A\} = \{b,d\}$
- $d^+(e) = 2$
- $d^-(e) = 2$
- $d(e) = 4$



Graphes orientés

Exercice 2

- Trouvez les degrés extérieurs et intérieurs de chacun des sommets du graphe ci-dessous



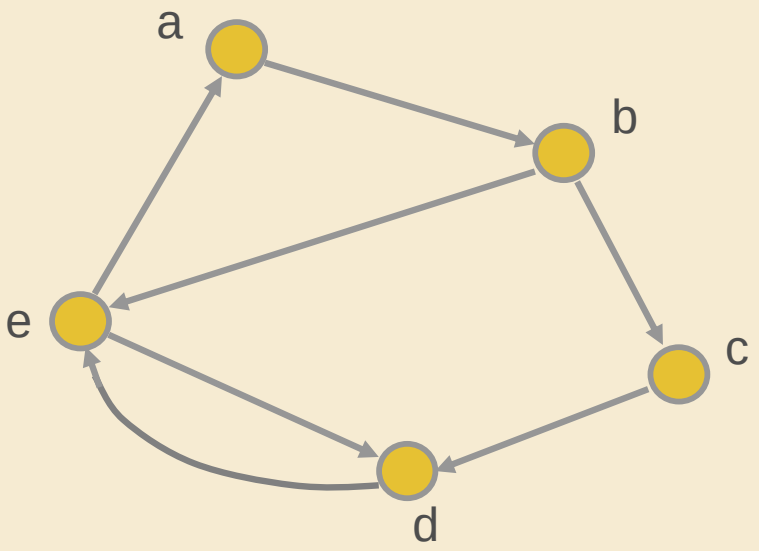
Graphes orientés

Chemin dans un graphe

- Soient u et v deux sommets distincts d'un graphe G=(V,A)
  - Un chemin de u à v dans G est une suite $u_0=u,u_1,\ldots,u_k=v$ de sommets 2 à 2 distincts tq $\forall i\in\{1,\ldots,k\}$ $(u_{i-1}, u_i)\in A$
  - u est l'origine du chemin / v est l'extrémité du chemin
  - La longueur d'un chemin est le nombre d'arcs qu'il possède

- Un chemin de longueur 4 dans
- le graphe reliant les
- sommets a à e

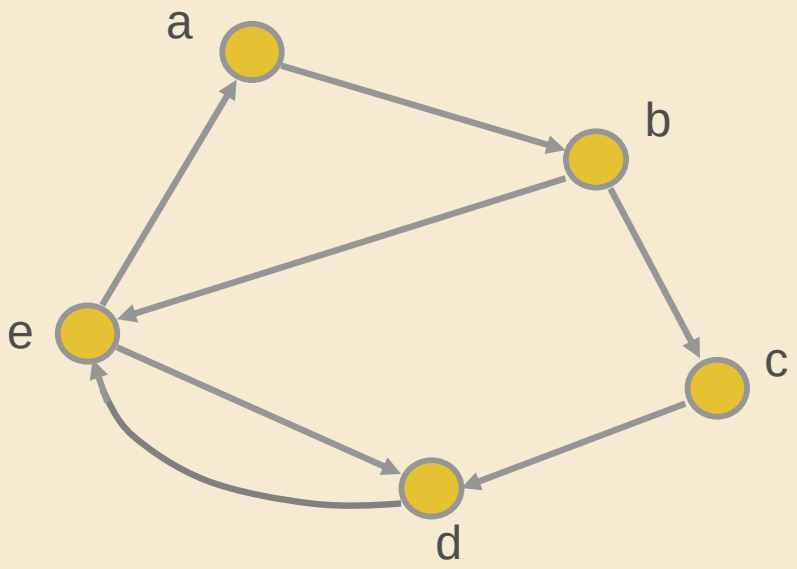
- ch=(a,b,c,d,e)



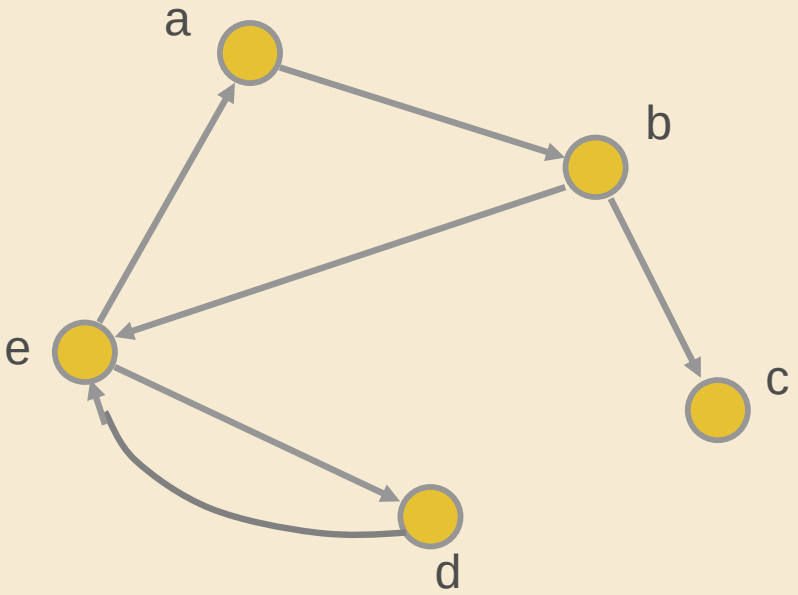
Graphes orientés

Graphe fortement connexe

- Un graphe orienté est fortement connexe si pour tout couple (u,v) de sommets distincts, il existe un chemin de u à v et un chemin de v à u



Graphe fortement connexe

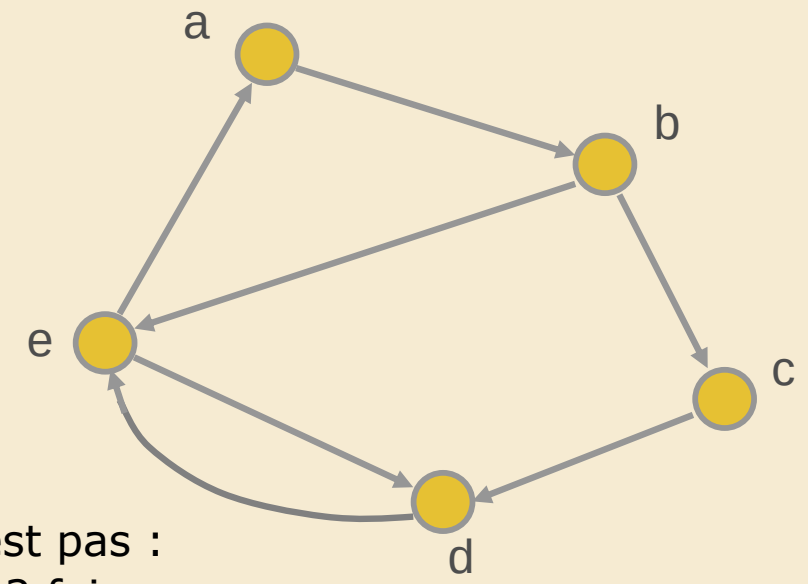


Graphe non fortement connexe

Graphes orientés

Chemin simple et circuit

- Une chemin est simple si chaque arc du chemin est empruntée une seule fois
- Un circuit est un chemin dont l'origine et l'extrémité sont confondues



- Les chemins (a,b,c,d) et
- (a,b,c,d,e,d) sont simples.
- Le chemin (a,b,c,d,e,d,e,d) ne l'est pas :
- le circuit (d,e,d,e) est emprunté 2 fois.
- Ce circuit pouvant être emprunté autant de fois que l'on veut, il y a un nombre infini de chemins de a à d

Graphes orientés

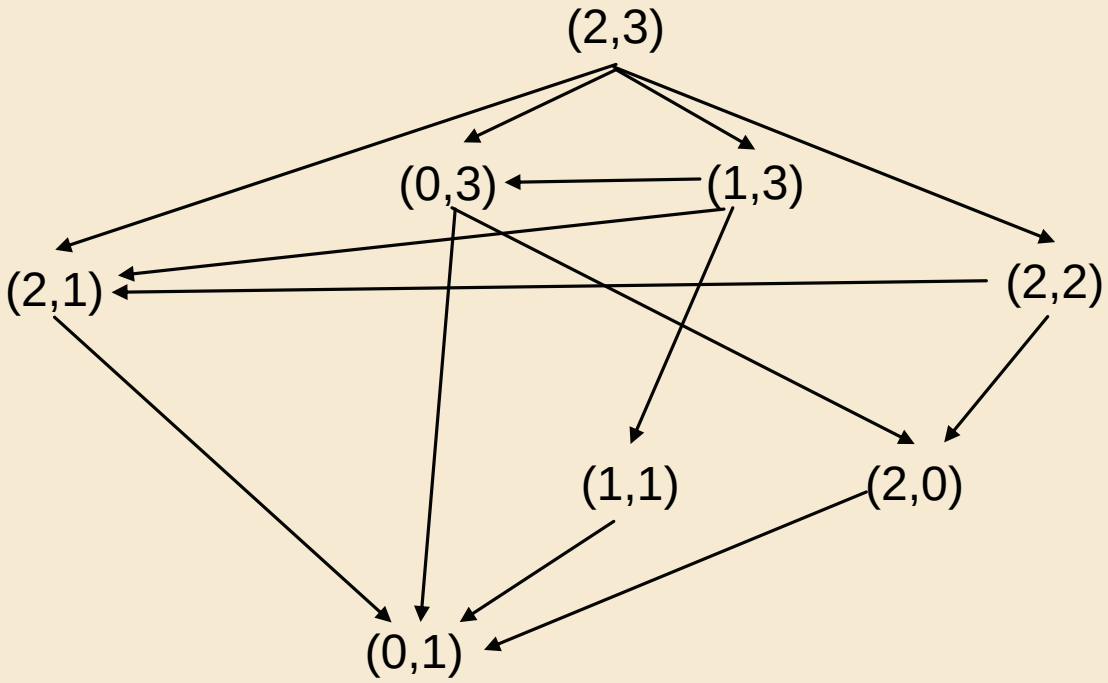
Exercice 3 : Jeu de Fan Tan

- Deux joueurs disposent de plusieurs tas d'allumettes. A tour de rôle, chaque joueur peut enlever un certain nombre d'allumettes de l'un des tas (selon la règle choisie). Le joueur qui retire la dernière allumette perd la partie.

- Modélisez ce jeu à l'aide d'un graphe dans le cas où l'on dispose au départ de deux tas contenant respectivement deux et trois allumettes, et où un joueur peut enlever une ou deux allumettes à chaque fois. Que doit jouer le premier joueur pour gagner la partie à coup sûr ?

Graphes orientés

Solution exercice 3 : Jeu de Fan Tan



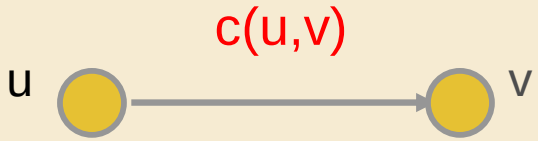
- Nous cherchons un chemin de longueur impaire
- Chemins gagnants : (2,3)-(1,3)-(X,Y)-(0,1) et (2,3)-(2,2)-(X,Y)-(0,1)

Graphes orientés valués

Définition

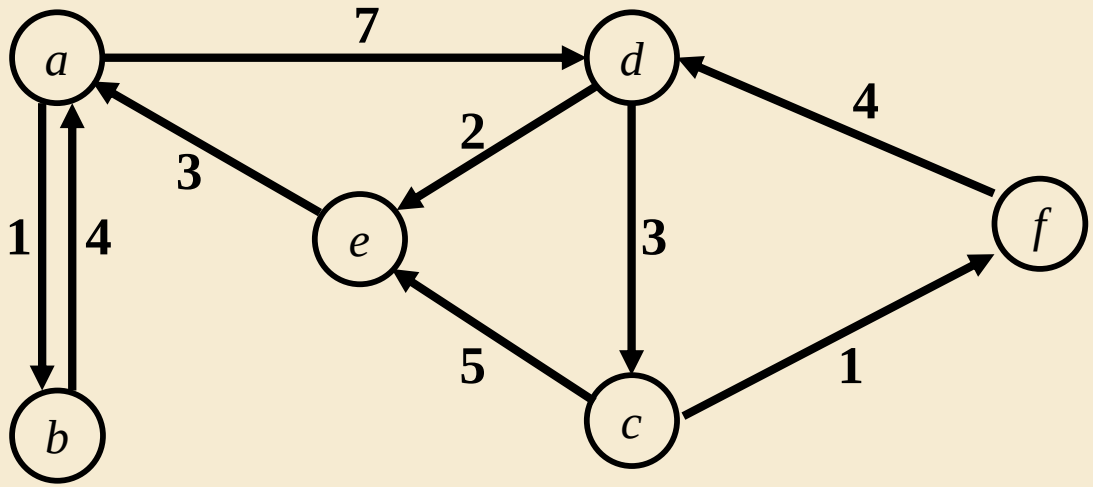
- Un graphe orienté valué *G* est défini par :
  - Un ensemble de *sommets* V
  - Un ensemble d'*arcs* A
  - Une valuation *C* : A $\rightarrow$ R qui à chaque arc du graphe associe une valeur réelle (coûts, poids, distances, …).

- On utilise alors la notation : *G* = (*V* , A, *C*)



Graphes orientés valués

Exemple

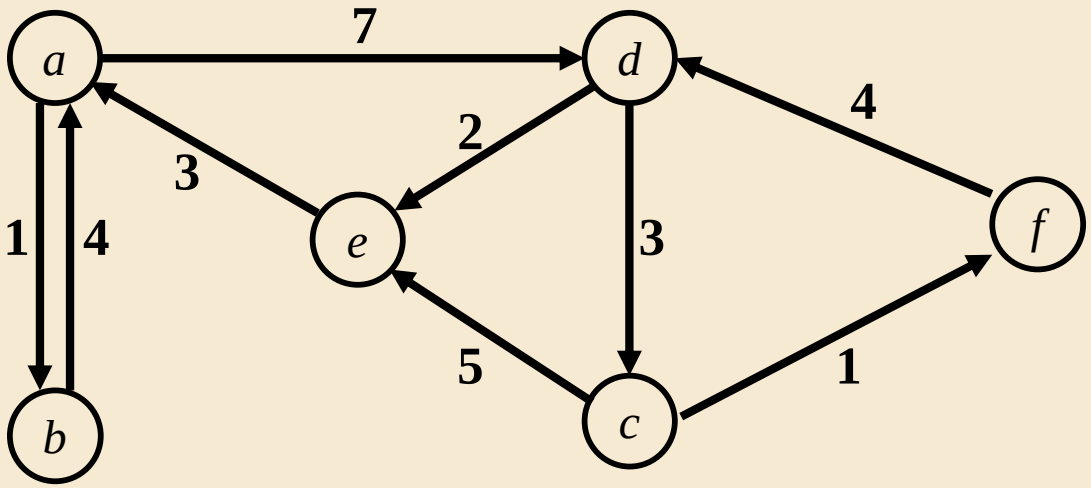


- V = {a, b, c, d, e, f}
- A = {(a,b),(a,d),(b,a),(c,e),(c,f),(d,c),(d,e),(e,a),(f,d)}
- c(a,b) = 1, c(b,a)= 4, c(d,e) = 2, c(a,d) = 7 …

Graphes orientés valués

Chemin dans un graphe valué

- le poids (ou longueur) *c(p)* d'un chemin *p* est la somme des poids des arêtes le long du chemin : c(p) = $\sum_{a \in p} c(a)$



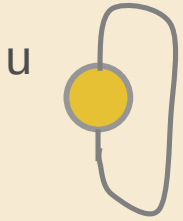
- Le chemin p = (a, d, c, f) est de poids 11
- Le chemin p’ = (a, d, e, a, d, c, f) est de poids 23

Graphes non orientés

Définition

- Un graphe non orienté G est composé :
  - V ensemble de sommets (ou nœuds)
  - E ensemble des arêtes
- $E \subseteq \{(u, v) : u, v \in V\}$
- (u, v) = (v, u) : relations symétriques

Une arête de u vers v

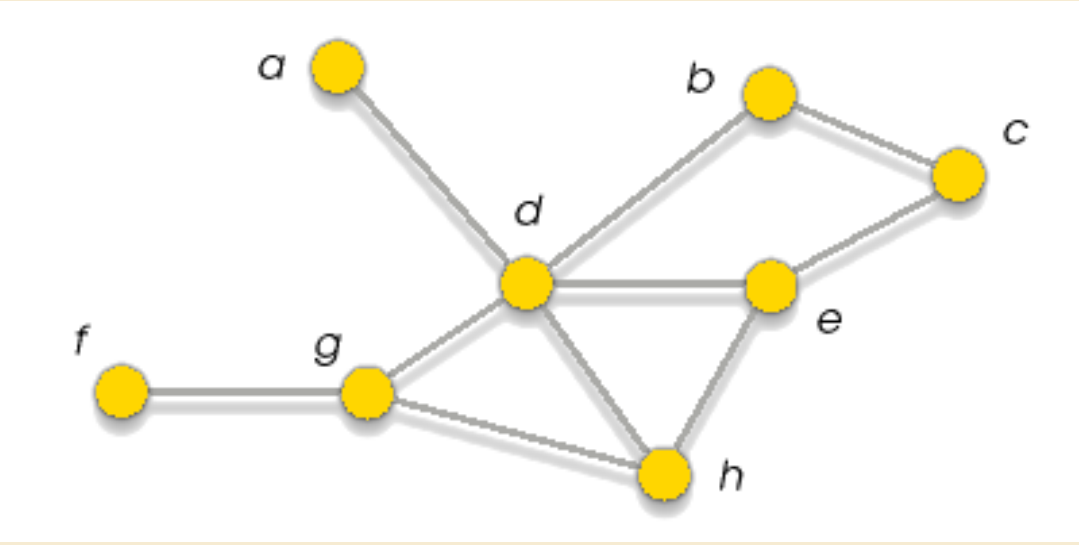


Une boucle

Graphes non orientés

Exemple

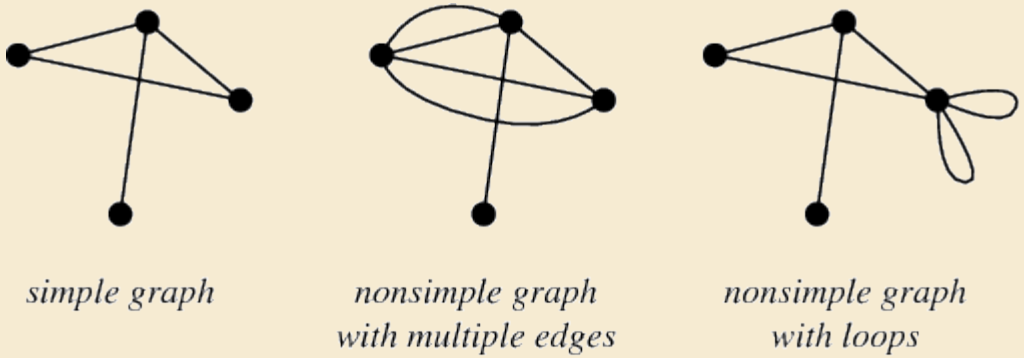
- Un graphe à 8 sommets, nommés *a* à *h*, comportant 10 arêtes.
- V={ a, b, c, d, e, f, g, h }
- E={(a,d),(b,c),(b,d),(d,e),(e,c),(e,h),(h,d),(f,g),(d,g),(g,h)}
- | V |=8 et | E |=10



Graphes simples

Graphes simples

- Un graphe est dit simple s'il n'a pas de liens doubles ni de boucles.
- Graphes non orientés : graphe sans boucle, et chaque couple de sommets est relié par au plus une arête.



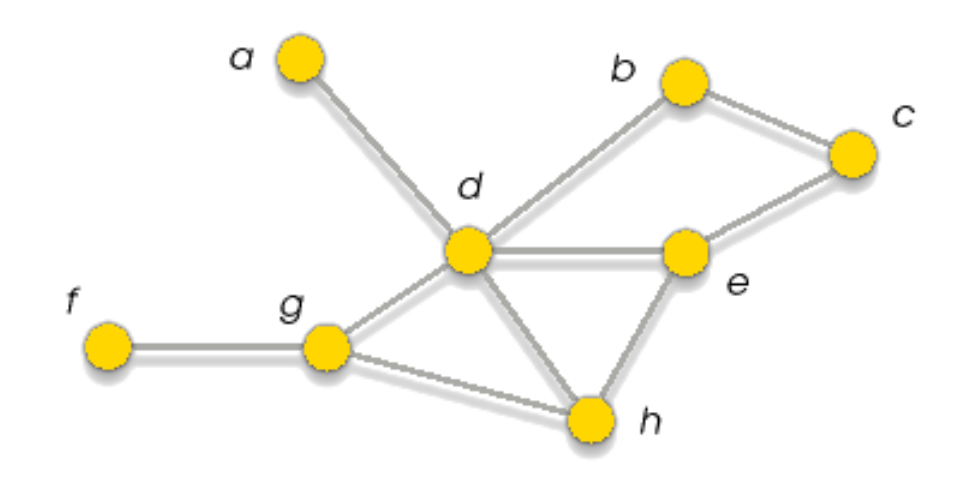
simple graph | *nonsimple graph with multiple edges* | *nonsimple graph with loops*

- Graphes orientés : graphe sans boucle, et chaque couple de sommets (u,v) est relié par au plus un arc de u vers v , et au plus par un arc de v vers u.

Graphes non orientés

Définition

- Soit G=(V,E) un graphe non orienté simple. Deux arêtes sont adjacentes si elles partagent une même extrémité

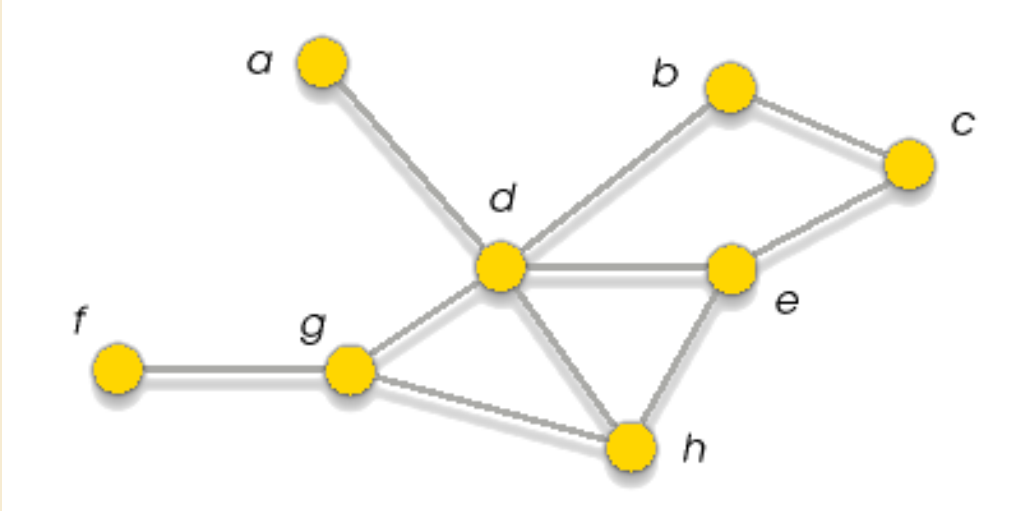


- e1=(a,d) et e2=(b,d) sont adjacentes
- e3=(f,g) et e4=(d,h) ne sont pas adjacentes

Graphes non orientés

Définition

- L'ensemble des voisins d'un sommet u dans un graphe G=(V,E) est N(u)={v∈V: (u,v) ∈ E}

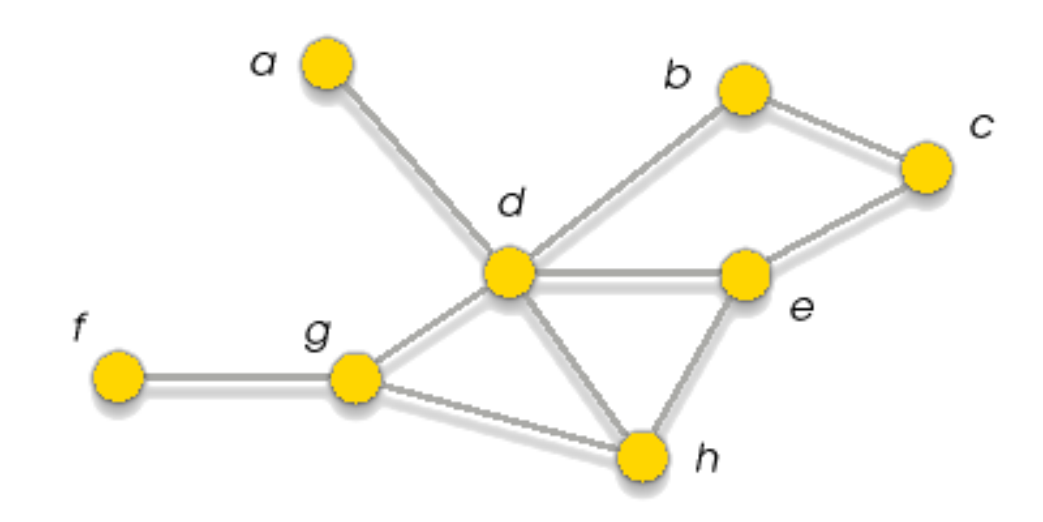


- N(a)={d}; N(g)={f,d,h}; N(d)={a,b,g,h,e} …

Graphes non orientés

Définition

- Une arête est incidente à un sommet *u* si *u* est l'une de ses extrémités.

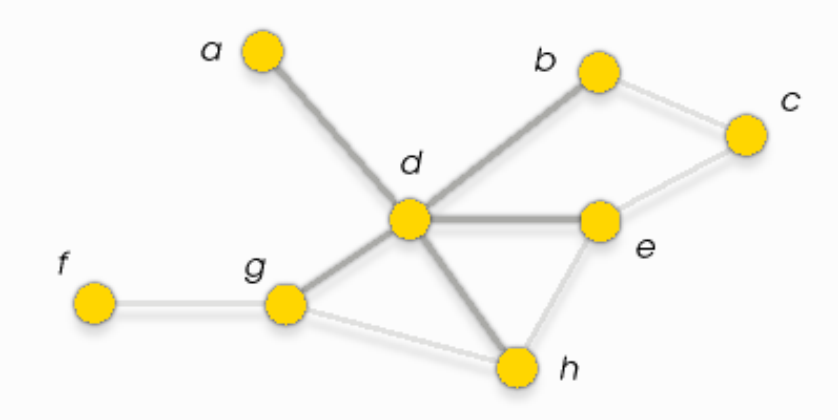


- L'arête e1=(h,g) est incidente au sommet h

Graphes non orientés

Définition

- Le degré d'un sommet u dans un graphe G=(V, E) est d(u)=|N(u)|
- Le degré minimum de G est δ=Min{d(u) : u ∈ V}
- Le degré maximum de G est Δ=Max{d(u) : u ∈ V}



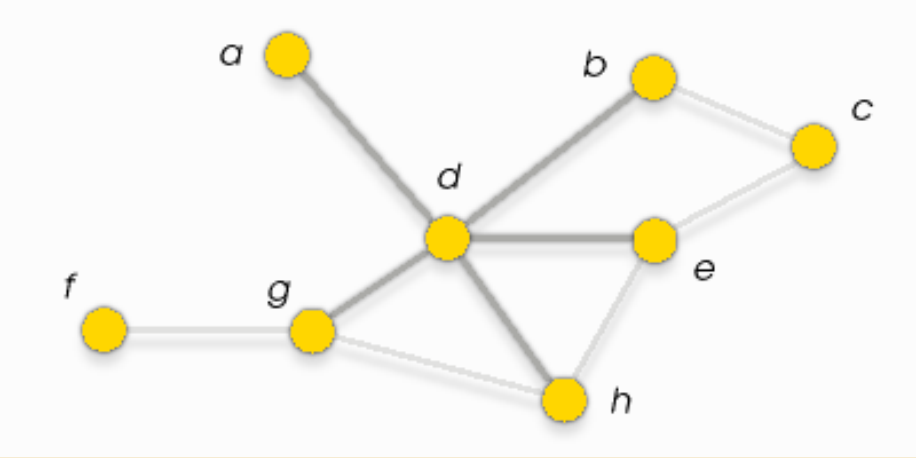
- d(d) = 5
- les arêtes incidentes à d
- sont (d,a),(d,b),(d,e),
- (d,h) et (d,g)

- δ=Min{d(u):u∈V} = 1 d(a)=1
- Δ=Max{d(u):u∈V} = 5 d(d) = 5

Graphes non orientés

Propriétés

- La somme des degrés des sommets d'un graphe est égal à 2 fois son nombre d'arêtes
- Le nombre de sommets de degré impair d'un graphe est pair

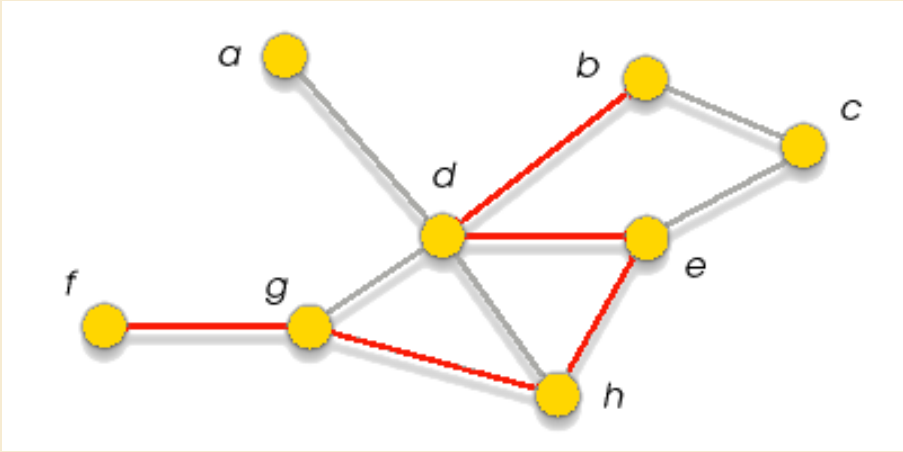


- Exercices :
  - Démontrer les propriétés précédentes

Graphes non orientés

Définitions

- Soient u et v deux sommets distincts d'un graphe G=(V,E)
  - Une chaîne de u à v dans G est une suite $u_0=u,u_1,\ldots,u_k=v$ de sommets 2 à 2 distincts tq $\forall i\in\{1,\ldots,k\}$ $(u_{i-1}, u_i)\in E$
  - u est l'origine de la chaîne / v est l'extrémité de la chaîne
  - La longueur d'une chaîne est le nombre d'arêtes qu'elle possède

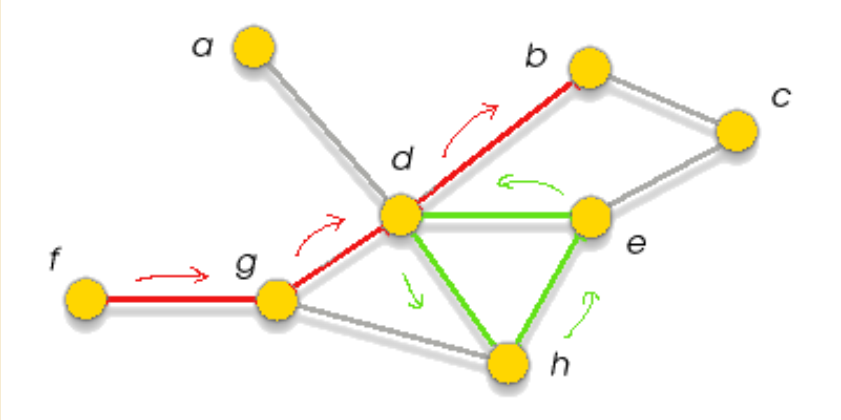


- p=(f, g, h, e, d, b)
- p est une chaîne de longueur 5 reliant les sommets f à b

Graphes non orientés

Remarque

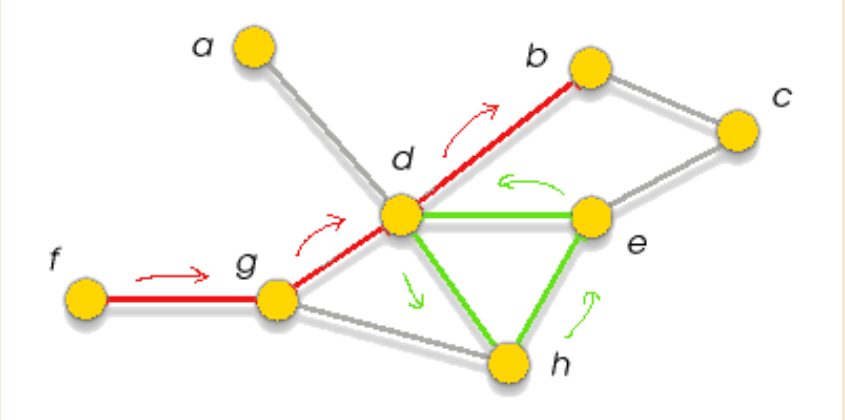
- Il existe d'autres chaînes pour aller de f à b
  - (f, g, d, b) de longueur 3,
  - (f, g, d, h, e, d, b) de longueur 6,
  - (f, g, d, h, e, d, h, e, d, b) de longueur 9,
  - ...



Graphes non orientés

Exemple

- Une chaîne est simple si chaque arête de la chaîne est empruntée une seule fois
- Un cycle est une chaîne dont l'origine et l'extrémité sont confondues



- Les chaînes (f, g, d, b)
- et (f, g, d, h, e, d, b)
- sont simples.

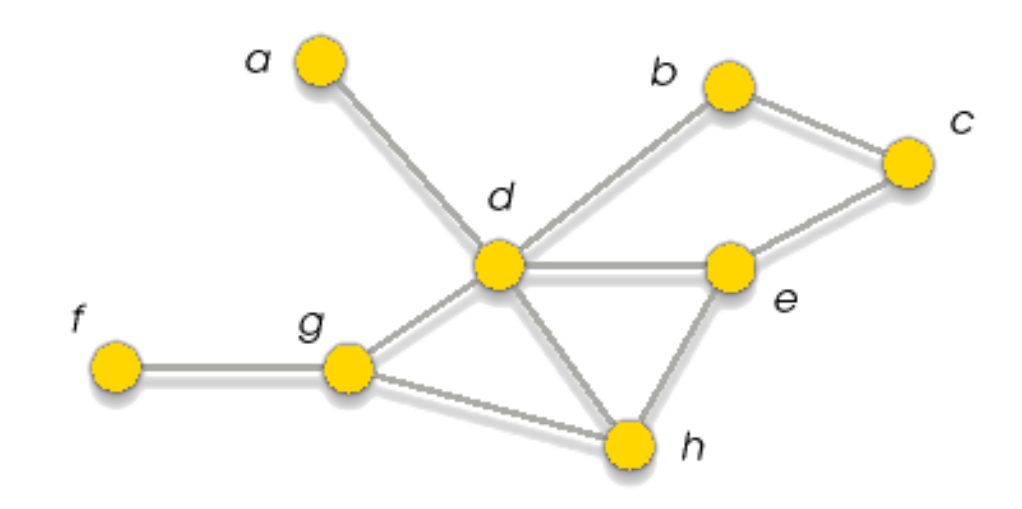
- La chaîne (f, g, d, h, e, d, h, e, d , b) n'est pas simple : le cycle (d, h, e, d) est emprunté 2 fois. Ce cycle pouvant être emprunté autant de fois que l'on veut, il y a un nombre infini de chaînes de f à b

GRAPHES PARTICULIERS

Graphes connexes

Définition

- Un graphe non-orienté est connexe si il existe une chaîne dans G entre toutes paires de sommets (distincts)

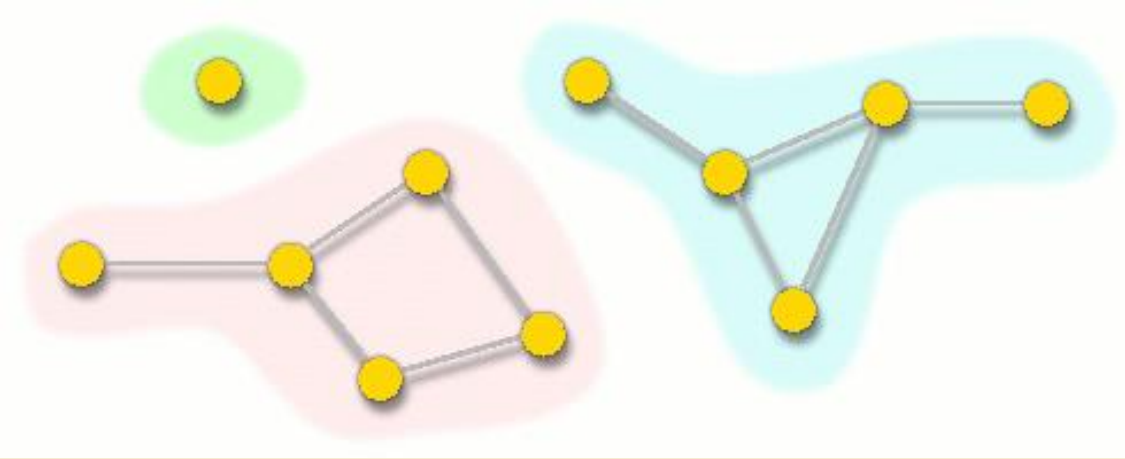


Ce graphe est connexe

Graphes non connexes

Définition

- Un graphe est non connexe si il existe un couple (u,v) de sommets distincts tq il n'existe pas de chaîne reliant le sommet u au sommet v



- Un graphe qui possède 3 composantes connexes, dont un sommet isolé

Graphes isomorphes

Définition

- Deux graphes orientés (resp. non orientés) G1 = (V1,E1) et G2 = (V2,E2) sont isomorphes ssi :
  - il existe une bijection f : V1 → V2 telle que :
  - (x, y) ∈ E1 ⇔ (f(x), f(y)) ∈ E2
  - (resp. {x, y} ∈ E1 ⇔ {f(x), f(y)} ∈ E2)

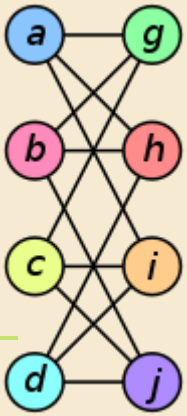
Complexité

- Le problème qui consiste à prouver que deux graphes sont isomorphes ou non, est un problème de la classe NP
- Décider de l'existence d'un isomorphisme entre deux arbres est dans la classe P

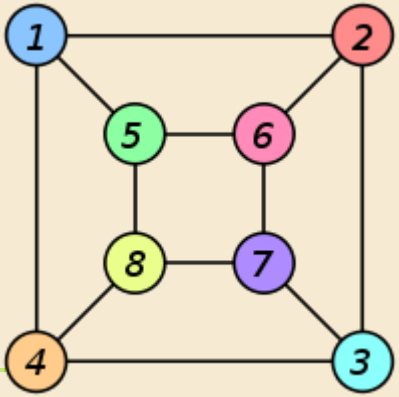
Graphes isomorphes

Exemple

Graphe G



Graphe H



Isomorphisme entre G et H

$f(a) = 1$
$f(b) = 6$
$f(c) = 8$
$f(d) = 3$
$f(g) = 5$
$f(h) = 2$
$f(i) = 4$
$f(j) = 7$

- On teste avant toute chose que les deux graphes on :
  - le même nombre de sommets
  - le même nombre d'arrêtes
  - le même nombre de sommets de degré d (pour tout d)

Graphes Eulériens

Définition (graphes non orientés)

- Dans un graphe non orienté, une chaine Eulérienne est une chaine qui emprunte une et une seule fois chaque arête du graphe.
- De même, un cycle Eulérien est un cycle qui emprunte une et une seule fois chaque arête du graphe.
- Enfin, un graphe comportant une chaine ou un cycle Eulérien est appelé graphe Eulérien.

Définition (graphes orientés)

- Dans un graphe orienté, un chemin Eulérien est un chemin qui emprunte une et une seule fois chaque arc du graphe.
- De même, un circuit Eulérien est un circuit qui emprunte une et une seule fois chaque arc du graphe.
- Enfin, un graphe comportant un chemin ou un circuit Eulérien est appelé graphe Eulérien.

Graphes Eulériens

Existence : cas de graphes non orientés

- Un graphe simple connexe admet un cycle Eulérien si et seulement s'il n'a pas de sommet de degré impair.
- Un graphe simple connexe admet une chaine Eulérienne entre deux sommets u et v si et seulement si le degré de u et le degré de v sont impairs, et les degrés de tous les autres sommets du graphe sont pairs.

Existence : cas de graphes orientés

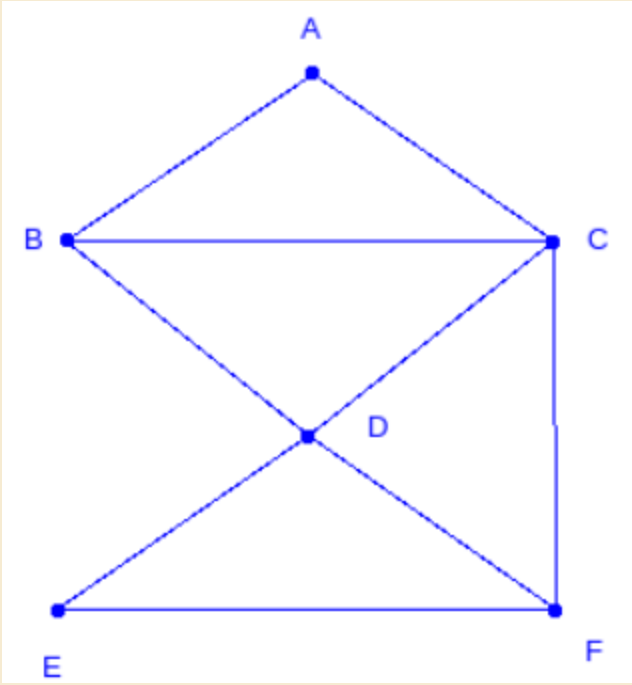
- Un graphe orienté fortement connexe admet un circuit Eulérien si et seulement si $d^+(u) = d^-(u)$ pour tout sommet $u \in V$.
- Un graphe orienté fortement connexe admet un chemin Eulérien de u vers v si et seulement si :
  - $d^+(u) = d^-(u) + 1$
  - $d^+(v) = d^-(v) - 1$
  - $d^+(s) = d^-(s)$ pour tout autre sommet $s \in V$.

Graphes Eulériens

Exemple : graphe non orienté

Sommets	A	B	C	D	E	F
Degrés	2	3	4	4	2	3

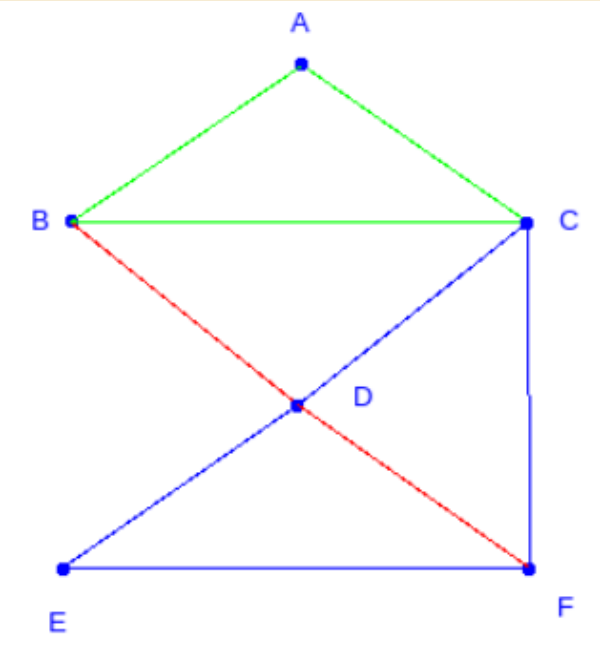
- Le graphe G est connexe
- Il a seulement deux sommets de
- degré impair : le sommet B et
- le sommet F.
- Il existe donc une chaîne
- Eulérienne entre les sommets
- B et F.



Graphes Eulériens

Algorithme d'Euler

- On choisit une chaîne d'origine B et d'extrémité F, ne contenant jamais deux fois le même arête. la chaîne B – D – F convient.
- On choisit un sommet de la chaîne
- précédente et, à partir de ce sommet,
- on adjoint un cycle ne contenant pas des
- arêtes déjà utilisées, on choisir le
- sommet B et le cycle B – A – C – B
- On obtient la chaîne B – A – C – B – D – F.
- On réitère l'étape précédente sur la chaîne
- obtenue jusqu'à avoir utilisé toutes les
- arêtes du graphe.
- La chaîne obtenue est par construction
- eulérienne, on choisit le sommet C et on
- adjoint le cycle C – D – E – F – C



- On obtient la chaîne B – A – C – D – E – F – C– B – D – F

Graphes Hamiltoniens

Définition (graphes non orientés)

- Une chaine Hamiltonienne est une chaine qui passe une et une seule fois par chacun des sommets du graphe.
- Un cycle Hamiltonien est un cycle qui passe une et une seule fois par chacun des sommets du graphe.
- Un graphe possédant un cycle Hamiltonien sera dit graphe Hamiltonien.

Définition (graphes orientés)

- Un chemin Hamiltonien est un chemin qui emprunte une et une seule fois chaque sommet du graphe.
- De même, un circuit Hamiltonien est un circuit qui emprunte une et une seule fois chaque sommet du graphe.
- Enfin, un graphe comportant un circuit Hamiltonien est appelé graphe Hamiltonien.

Graphes Hamiltoniens

Chemin Hamiltonien

- Le problème du chemin Hamiltonien consiste à :
  - trouver une chaîne Hamiltonienne ou un cycle Hamiltonien dans un graphe non orienté donné
  - ou à trouver un chemin Hamiltonien ou un circuit Hamiltonien dans un graphe orienté donné.

Condition d'existence

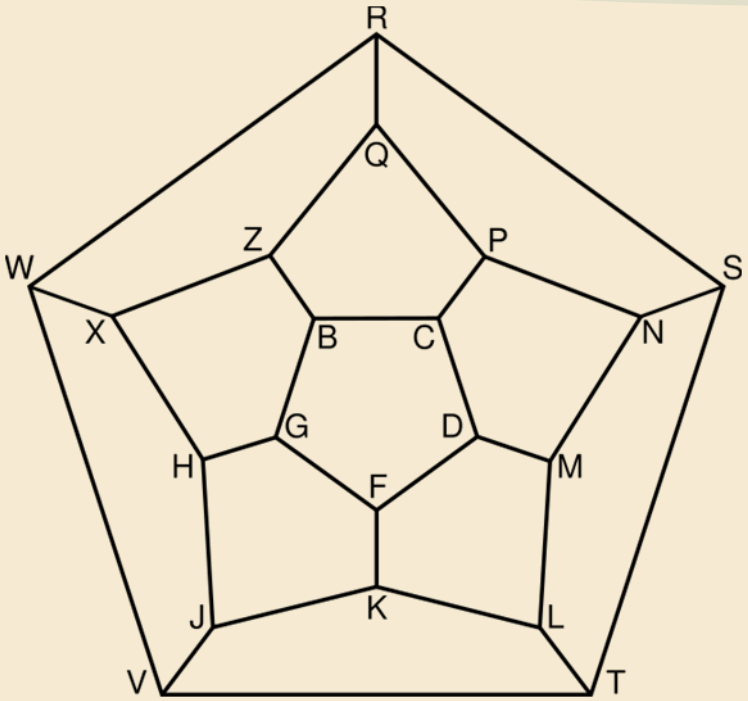
- Jusqu'à présent, on ne connait aucune condition nécessaire et suffisante d'existence de cycles (chaines, circuits ou chemins) Hamiltoniens, valable pour tous les graphes

Complexité

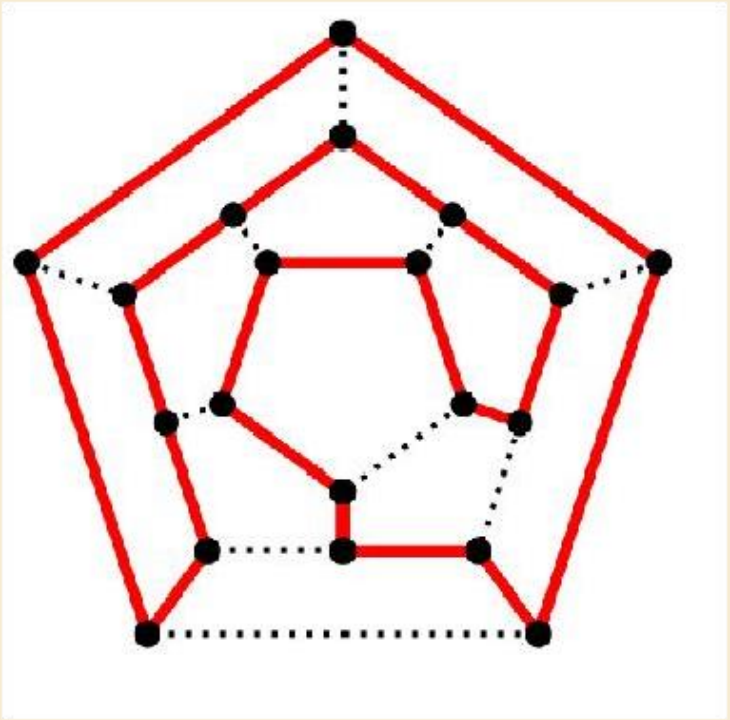
- Ce problème est un des premiers à avoir été montré comme étant NP-complet.

Graphes Hamiltoniens

Exemple :



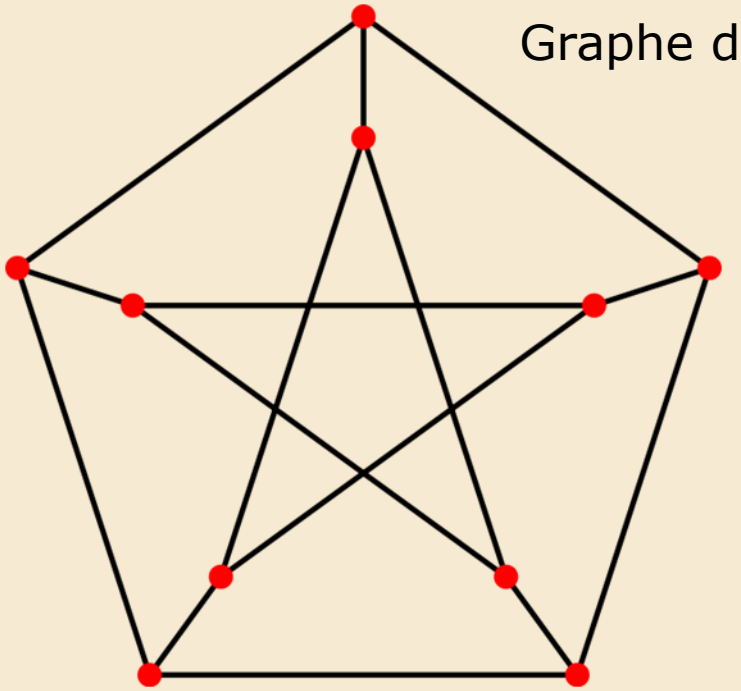
Graphe de Hamilton



Graphe Hamiltonien

Graphes Hamiltoniens

Exemple :



Graphe de Petersen

Graphe de Petersen : graphe non Hamiltonien

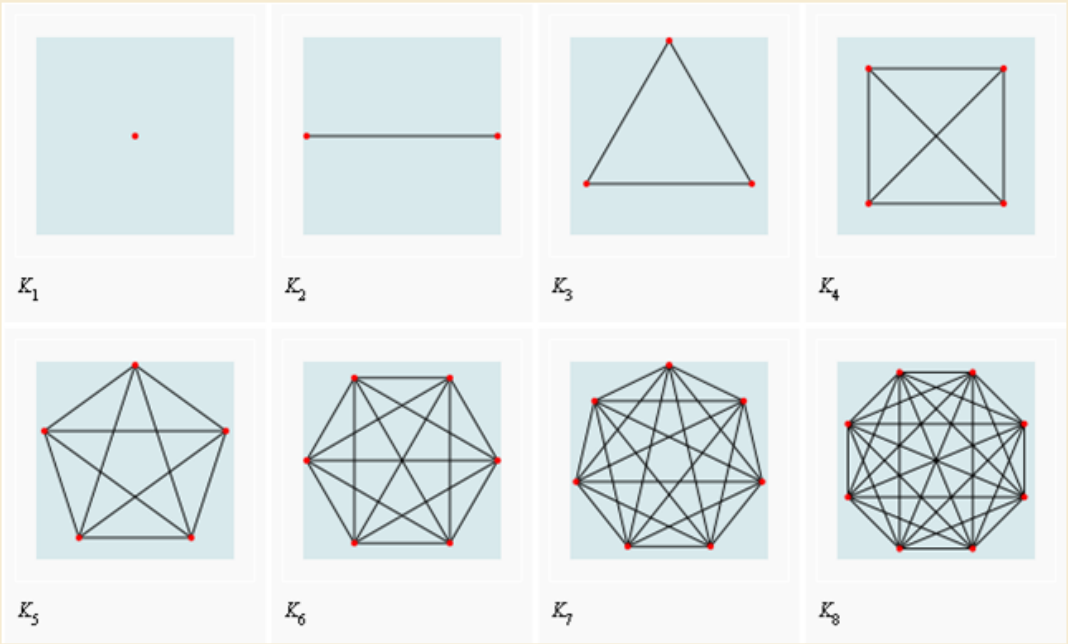
Ce graphe ne possède pas de cycle Hamiltonien

Par contre, il possède 240 chemins Hamiltoniens

Graphes complets

Définition (graphes non orientés)

- Un graphe simple non orienté est dit complet si et seulement si tous ses sommets sont reliés deux à deux par une arête.
- Un graphe complet d'ordre n est noté : $\mathbf{K_n}$

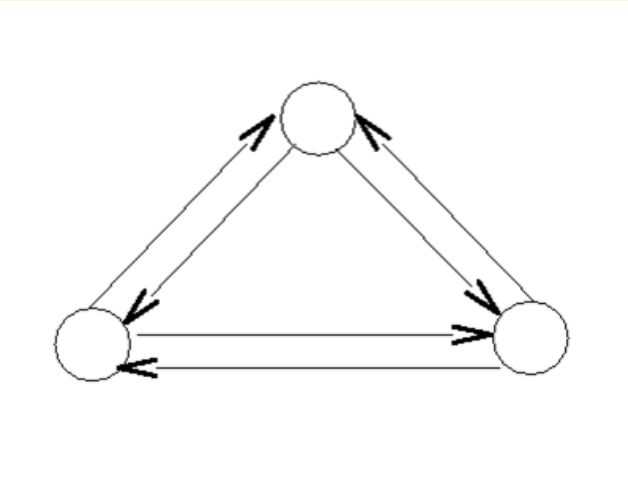


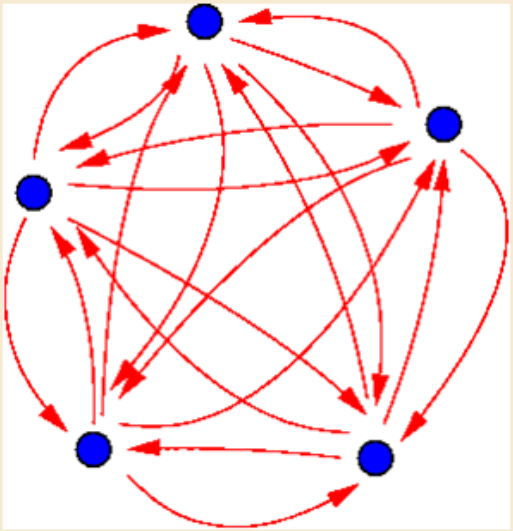
- Le nombre d'arêtes d'un graphe complet d'ordre n est : n(n-1)/2

Graphes complets

Définition (graphes orientés)

- Un graphe simple orienté est dit complet si et seulement si il comporte un arc (u,v) et un arc (v,u) pour tous couple de sommets distincts u, v.
- Un graphe complet d'ordre n est noté : $\mathbf{K_n}$



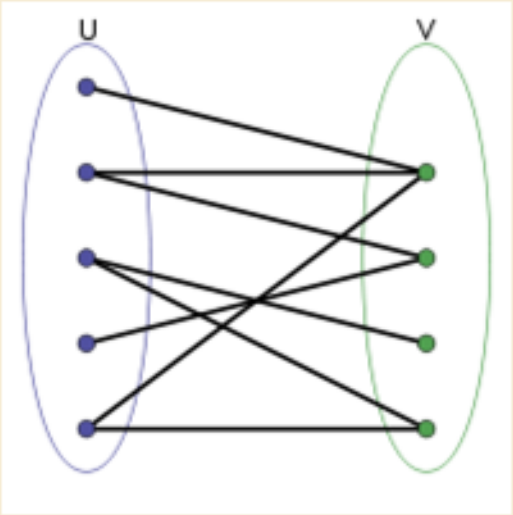


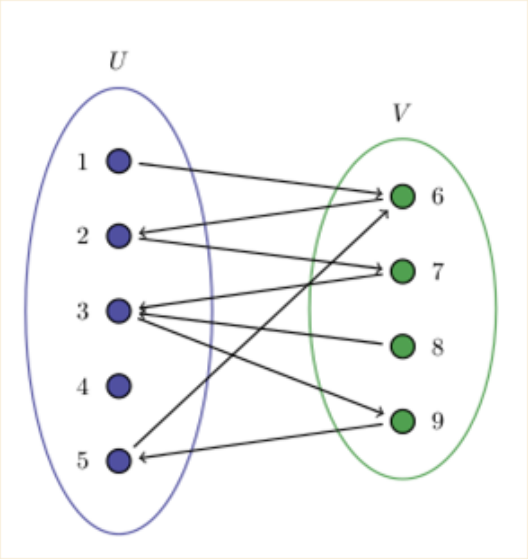
- Le nombre d'arcs d'un graphe complet d'ordre n est : **n(n-1)**

Graphes bipartis

Définition (graphes orientés ou non)

- Un graphe simple est dit biparti si et seulement si il existe une partition de son ensemble de sommets en deux sous-ensembles U et V telle que chaque arête/arc ait une extrémité dans U et l'autre dans V.



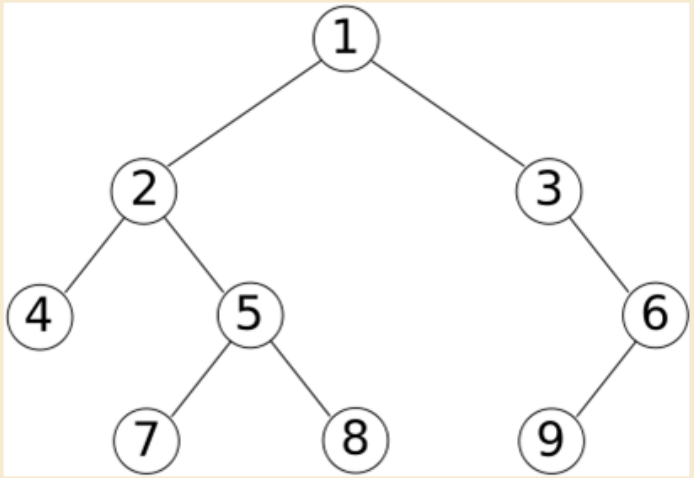


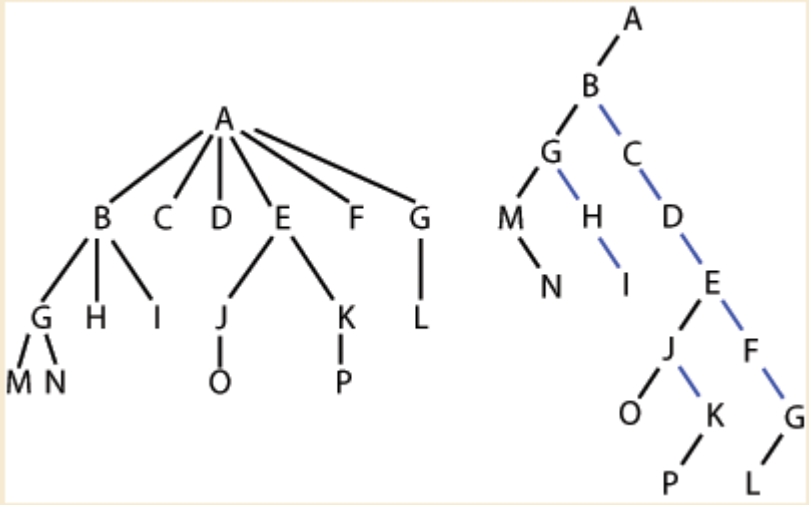
- Un graphe est biparti si et seulement s'il ne contient pas de cycle impair

Arbres

Définition

- Un arbre (tree) est un graphe non orienté connexe et sans cycle.





- On distingue deux types de sommets dans un arbre :
  - Les feuilles dont le degré est 1 ;
  - Les nœuds internes dont le degré est supérieur à 1.

Arbres

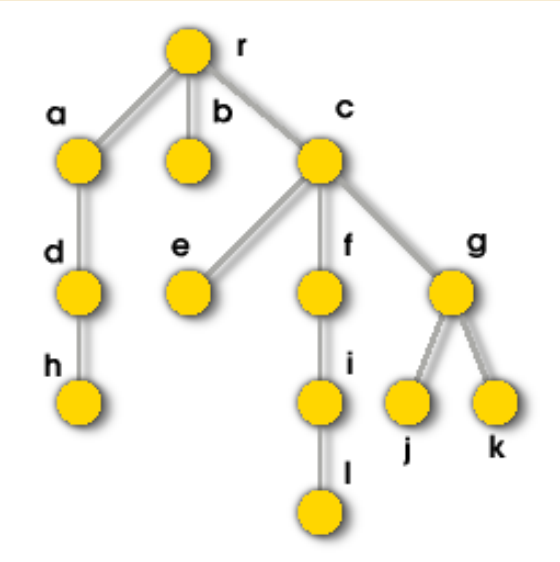
Vocabulaires

- Soit T un arbre de racine r
  - Le père d'un sommet (ou nœud) x est l'unique voisin de x sur le chemin de la racine à x. La racine r est le seul sommet sans père.
  - Les fils d'un sommet x sont les voisins de x autres que son père.
  - Une feuille est un sommet sans fils. Les feuilles correspondent aux sommets de degré 1.
  - La hauteur h(T) de l'arbre T est la longueur du plus long chemin de la racine à une feuille.

Arbres

Exemple

- On représente habituellement un graphe de racine r par niveaux, à la façon d'un arbre généalogique. Au niveau 0 apparaît la racine, au niveau 1 ses fils, au niveau 2 les fils de ses fils, etc... Le niveau d'un sommet correspond à sa profondeur dans l'arbre, c'est à dire la longueur de son chemin à la racine.



Dans l'exemple :

- L'arbre a une hauteur de 4.
- Les feuilles de l'arbres sont *h, b, e, l, j, k*.
- Les fils de la racine sont *a, b, c*.
- Le père de *g* est *c*

Arbres

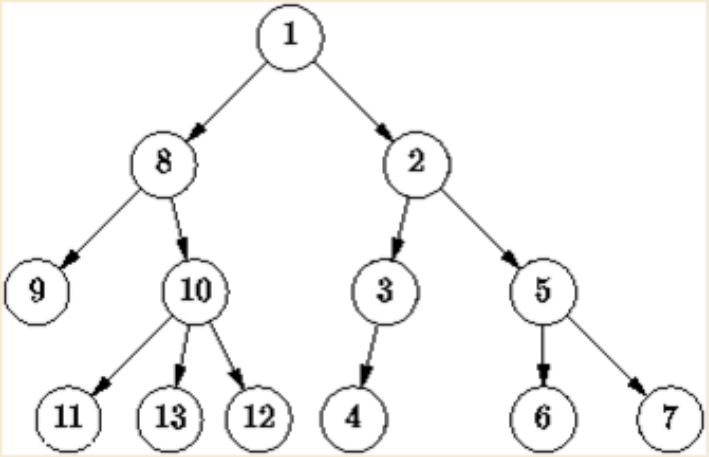
Théorème

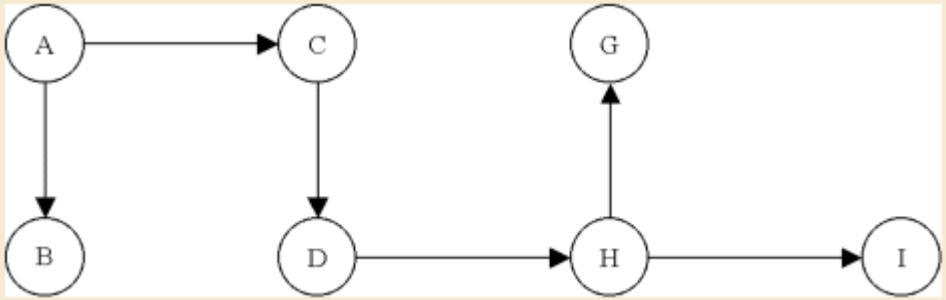
- Soit T=(V,E) un graphe à n sommets.
- Les propriétés suivantes sont équivalentes :
  - T est un arbre
  - T est sans cycle et admet n-1 arêtes
  - T est connexe et admet n-1 arêtes
  - T est sans cycle et en ajoutant une arête on crée un cycle et 1 seul
  - T est connexe et si on supprime une arête quelconque, il n'est plus connexe

Arborescences

Définition

- Une arborescence est un graphe orienté sans circuit admettant une racine r telle que, pour tout autre sommet s de S, il existe un chemin unique allant de r vers s.

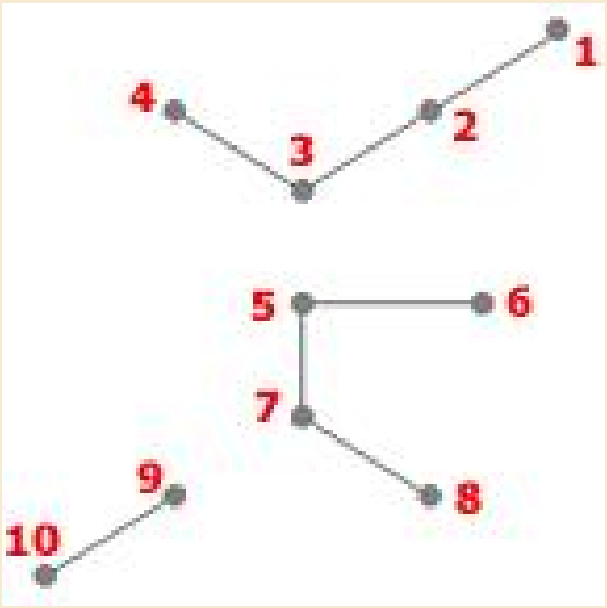


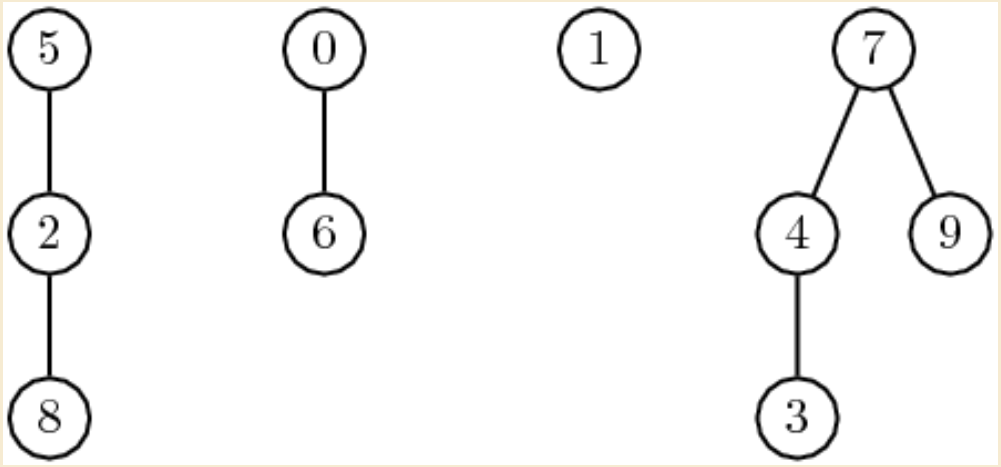


Forêts

Définition

- Une forêt est un graphe dont les composantes connexes sont des arbres

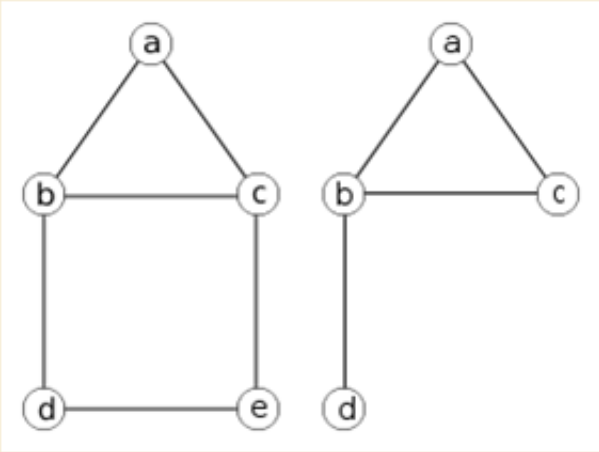


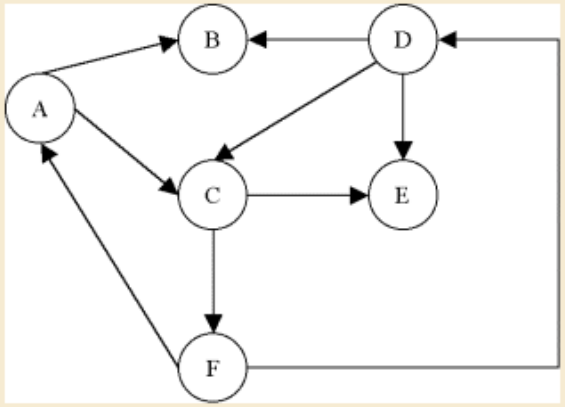


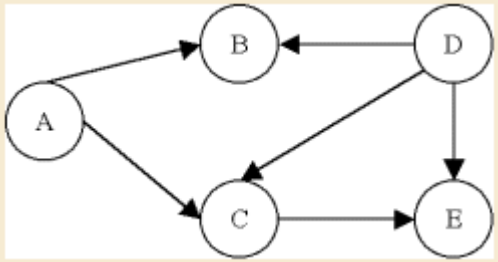
Sous-graphes

Définition

- Un sous-graphe (subgraph) d'un graphe orienté ou non est le graphe obtenu en supprimant certains sommets et tous les arcs ou arêtes incidents aux sommets supprimés.





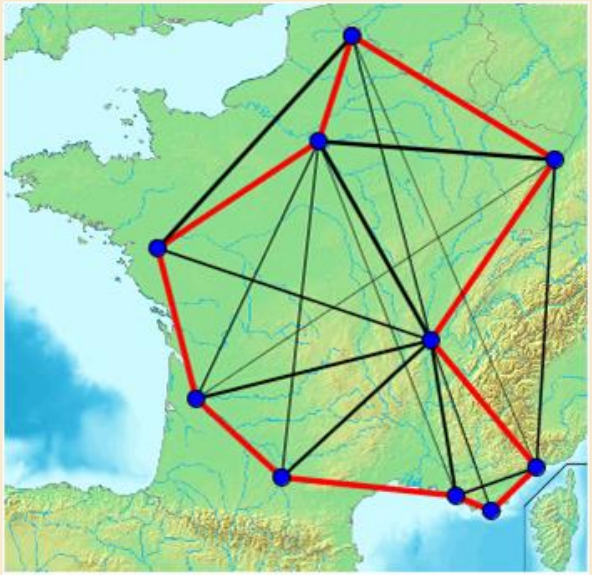


PROBLEMES PARTICULIERS

Problème du voyageur de commerce

Introduction

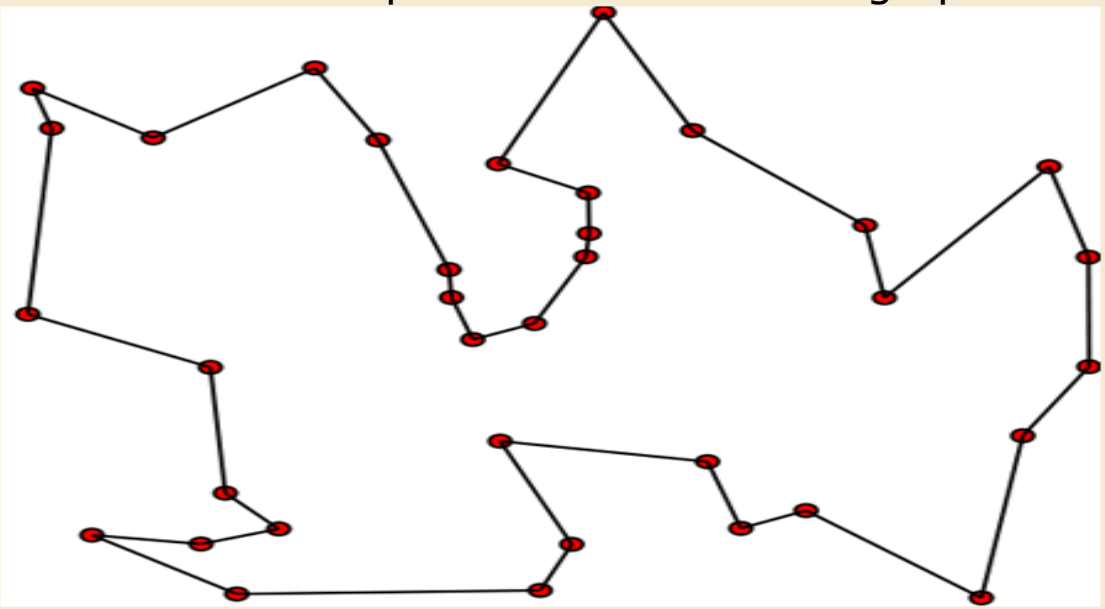
- L'énoncé du problème du voyageur de commerce est le suivant : étant donné n villes et les distances séparant chaque point, trouver un circuit de longueur totale minimale qui passe exactement une fois par chaque ville.



Problème du voyageur de commerce

Définition

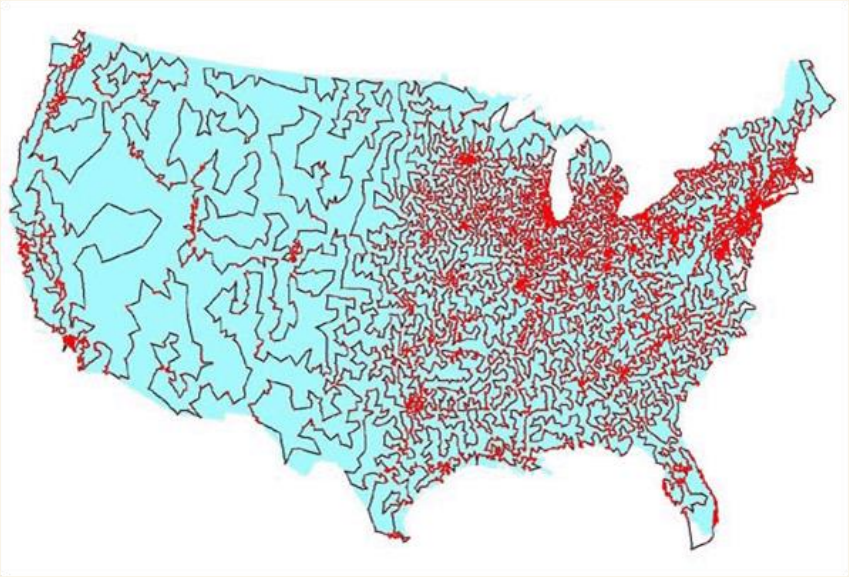
- Soit un graphe complet G = (V, E, W)
  - V un ensemble de sommets,
  - E un ensemble d'arêtes
  - W une fonction de coût sur les arêtes.
- Le problème du voyageur de commerce revient à trouver un plus cycle Hamiltonien de poids minimal dans le graphe G

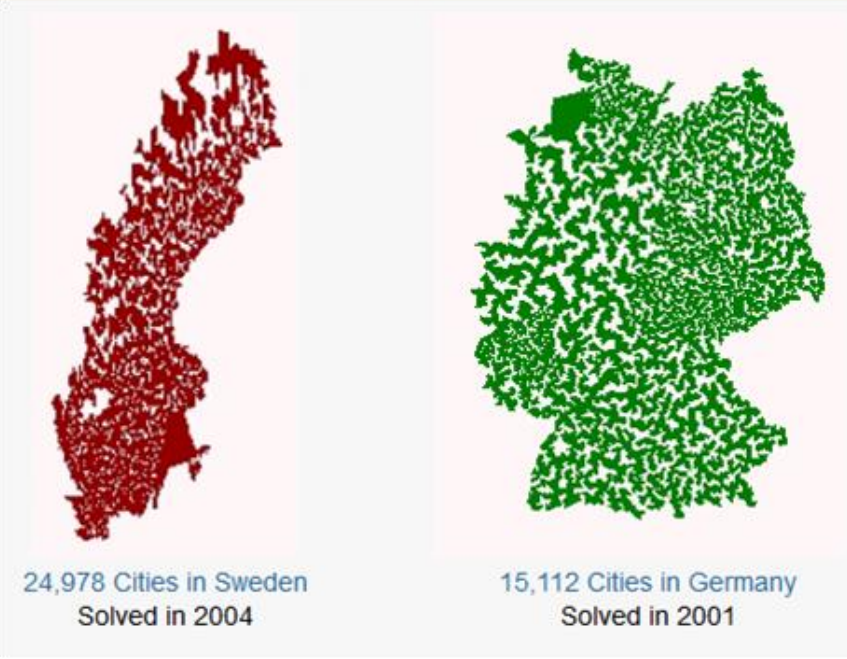


Problème du voyageur de commerce

Résolution

- 1954 : solution pour 49 villes par Dantzig, Fulkerson et Johnson
- 1975 : solution pour 100 villes par Camerini, Fratta and Maffioli
- 1987 : solution pour 532, puis 2392 villes par Padberg et Rinaldi
- 1998 : solution pour les 13 509 villes des Etats-Unis.
- 2001 : solution pour les 15 112 villes d'Allemagne
- 2004 : solution pour les 24 978 villes de Suède





Problème du voyageur de commerce

Heuristique du plus proche voisin

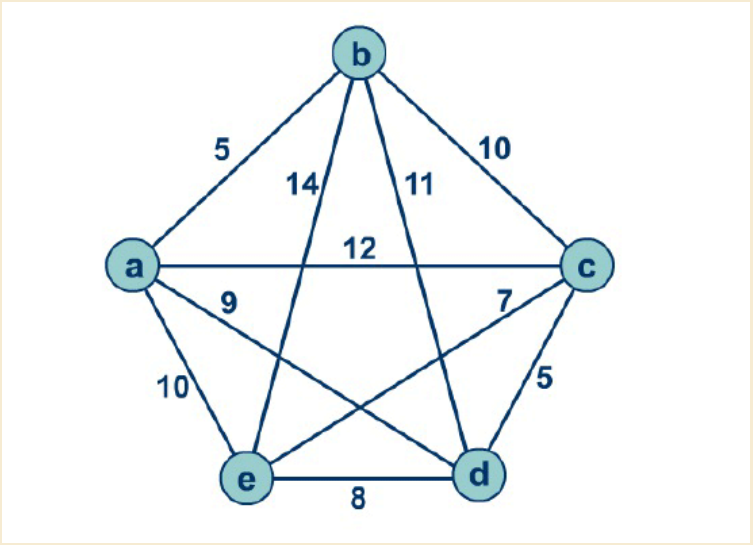
- **Principe :** On part d'une ville quelconque et l'on se dirige vers la ville la plus proche sans repasser par une ville déjà visitée.

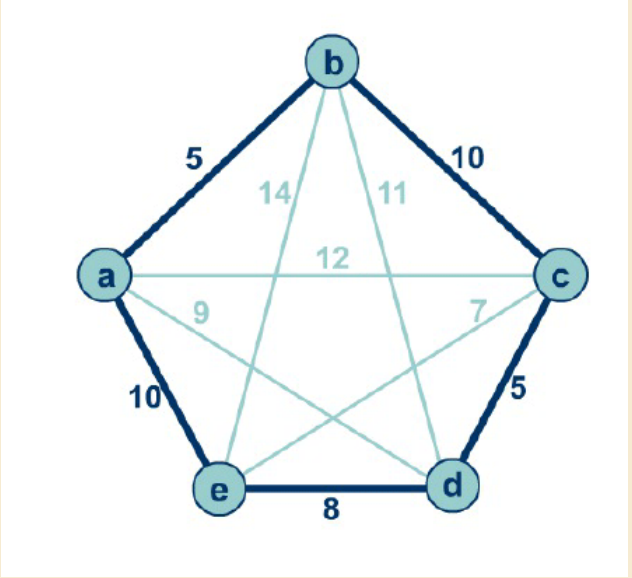
- **Algorithme**
  - Partir d'un sommet
  - **TANT QUE** la tournée n'est pas complète
    - Aller à la ville la plus proche non encore visitée
  - **FINTANTQUE**

Problème du voyageur de commerce

Heuristique du plus proche voisin

- **Algorithme**
  - Partir d'un sommet
  - TANT QUE la tournée n'est pas complète
    - Aller à la ville la plus proche non encore
    - visitée
  - FINTANTQUE





Si on part de a, la ville la plus proche est b, puis c, puis d, puis e, puis a.

On obtient la tournée a b c d e a de longueur : 5 + 10 + 5 + 8 + 10 = 38

Problème du voyageur de commerce

Heuristique de liste

- **Principe :**
- On commence par trier les arêtes par longueur croissante.
- Puis on parcourt la liste triée des arêtes et on sélectionne les arêtes de manière à ne pas créer de circuit, ni de sommet de degré 3, c'est-à-dire une fourche, ceci jusqu'à ce qu'on puisse revenir au sommet de départ.

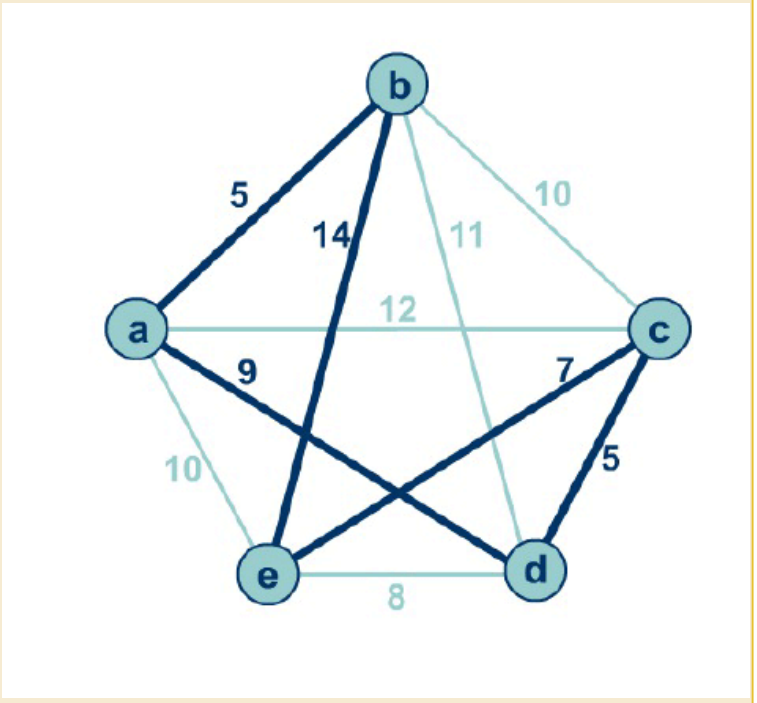
- **Algorithme :**
- Trier les arêtes par ordre de longueur croissante
- **TANTQUE** la tournée n'est pas complète
  - Prendre la première arête non examinée de la liste
  - Si elle ne crée ni cycle, ni fourche avec les arêtes précédemment sélectionnées, la retenir.
- **FINTANTQUE**

Problème du voyageur de commerce

Heuristique de liste

Algorithme :
Trier les arêtes par ordre de longueur croissante
TANTQUE la tournée n'est pas complète
    Prendre la première arête non examinée de la liste
    Si elle ne crée ni cycle, ni fourche avec les arêtes précédemment sélectionnées, la retenir.
FINTANTQUE

On commence par (a, b) de longueur 5 puis (c, d) de longueur 5 également, puis(c, e) de longueur 7.
On ne peut pas prendre (e, d) de longueur 8 qui créerait un sous-cycle decd, donc on prend (a, d) de longueur 9 et on termine par (b, e) de longueur 14.
On trouve la tournée a b e c d a de longueur 40

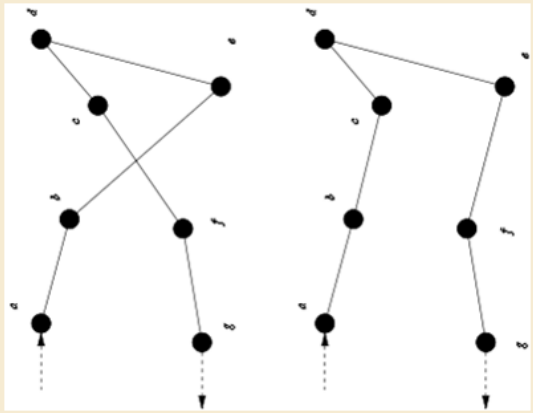


Problème du voyageur de commerce

Animation

- Étant donné un ensemble de 200 villes, quatre algorithmes sont utilisés pour trouver un plus court cycle passant une et une seule fois par ces 200 villes.
  - **Chemin au hasard :** sélectionner au hasard la prochaine ville parmi les villes restantes non encore visitées, jusqu'à ce que toutes les villes sont visitées.
  - **Greedy :** sélectionner la prochaine ville non encore visitées qui est la plus proche de la ville actuelle (i.e. plus proche voisin)
  - **2-Opt :** créer d'abord une solution initiale, puis l'optimiser avec l'algorithme 2-opt

2-opt : à chaque étape, on supprime deux arêtes de la solution courante et on reconnecte les deux tours ainsi formés

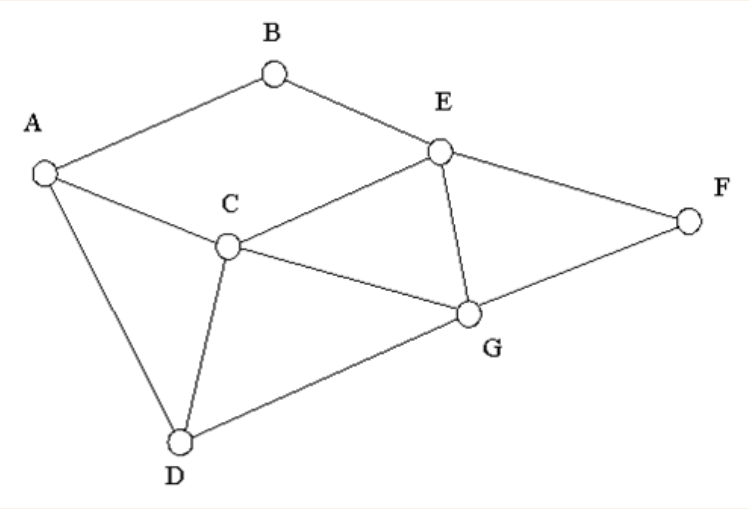


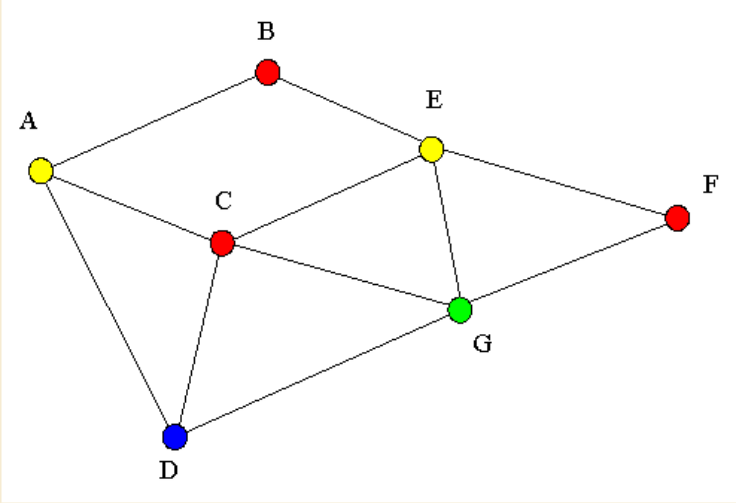
- **Recuit simulé :** Créer d'abord une solution initiale, puis l'optimiser avec **2-opt** en combinaison avec **recuit simulé**.

Coloriage des sommets d'un graphe

Définition

- Le problème de coloriage des sommets d'un graphe consiste à affecter à chaque sommet une couleur telle que 2 sommets adjacents n'aient pas la même couleur.
- On cherche à minimiser le nombre de couleurs utilisées. Le plus petit nombre de couleurs permettant la coloration est appelée nombre chromatique du graphe.





- Le problème de k-coloriage est NP. C'est encore un problème à un million de dollars !

Coloriage des sommets d'un graphe

Algorithme glouton

- **Principe :**
  - L'heuristique la plus simple consiste à prendre les sommets dans un ordre donné (qui peut être quelconque) et à leur affecter la première couleur possible.

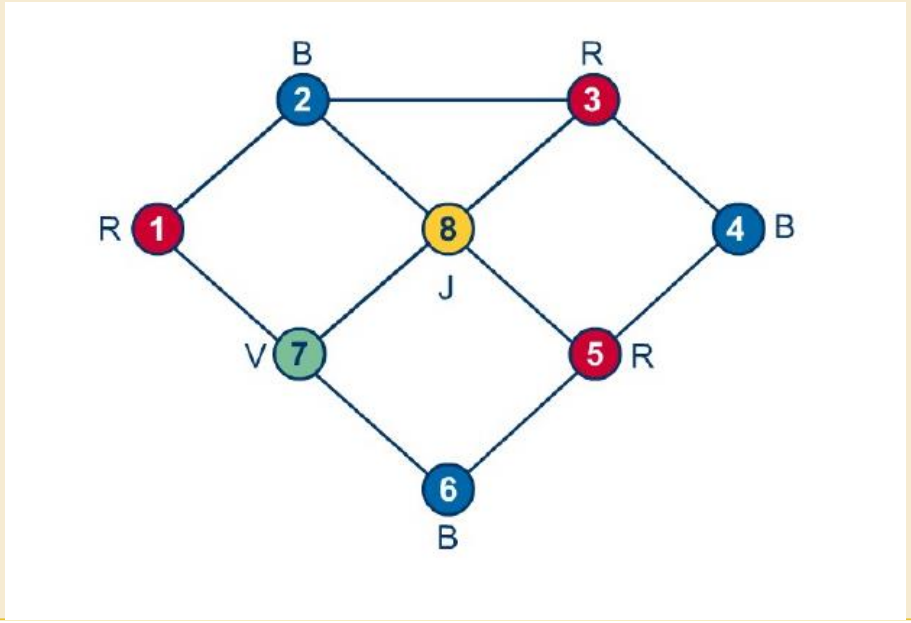
- **Algorithme :**
  - Numéroter les sommets de 1 à n
  - Numéroter les couleurs
  - **POUR** i de 1 à n
    - Colorier le sommet i avec la première couleur possible.
  - **FINPOUR**

- La solution dépend de l'ordre dans lequel les sommets seront examinés.

Coloriage des sommets d'un graphe

Exemple

Ordre des couleurs : 1 : Rouge, 2 : bleu, 3 : vert, 4 : jaune
On commence par le sommet 1 à qui l'on attribue la première couleur : rouge.
La première couleur possible pour le sommet 2 est la couleur 2, le bleu.
Pour le sommet 3, la couleur possible de plus petit numéro est le rouge (couleur 1).
On peut ensuite mettre le sommet 4 en bleu, 5 en rouge et 6 en bleu.
Pour le sommet 7 on doit utiliser le vert et pour le 8 la quatrième couleur, le jaune.
On vient d'obtenir une coloration avec 4 couleurs. Mais ce n'est pas la solution optimale



Coloriage des sommets d'un graphe

Algorithme glouton amélioré

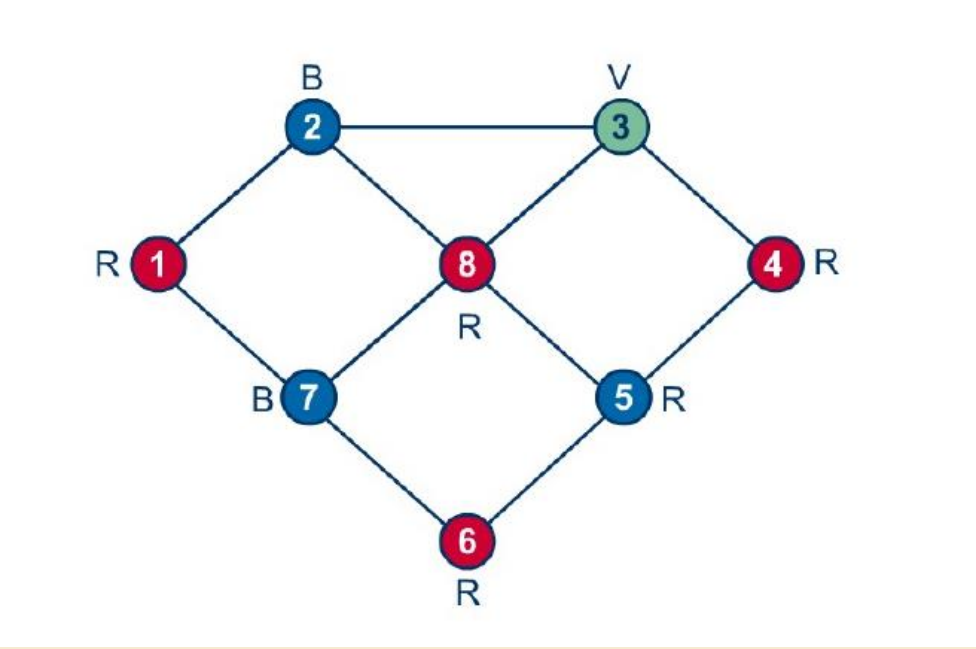
- **Principe :**
  - L'amélioration de l'heuristique précédente passe par un choix plus judicieux de l'ordre d'examen des sommets.
  - On modifie l'heuristique précédente en numérotant les sommets dans l'ordre de leur degré décroissant. Pour le reste, le principe est inchangé.

- **Algorithme :**
  - Numéroter les sommets dans l'ordre des degrés décroissants
  - Numéroter les couleurs
  - **POUR** i de 1 à n
    - Colorier le sommet i avec la première couleur possible.
  - **FINPOUR**

Coloriage des sommets d'un graphe

Exemple

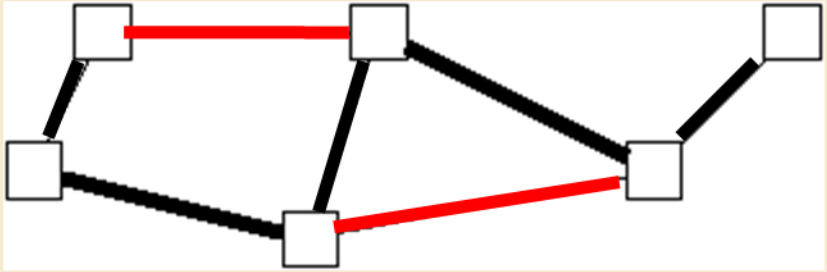
Ordre des couleurs : 1 : Rouge, 2 : bleu, 3 : vert, 4 : jaune
Ordre des sommets : 8, 2, 3, 5, 7, 1, 4, 6
On commence donc la coloration par le sommet qui porte maintenant le numéro 8 car il est de degré 4, on passe ensuite au sommet 2 (de degré 3) puis au sommet 3 etc.…
On constate que 3 couleurs suffisent maintenant et l'on ne peut trouver mieux en raison des sommets 2, 3 et 8 qui doivent être 2 à 2 de couleurs différentes.



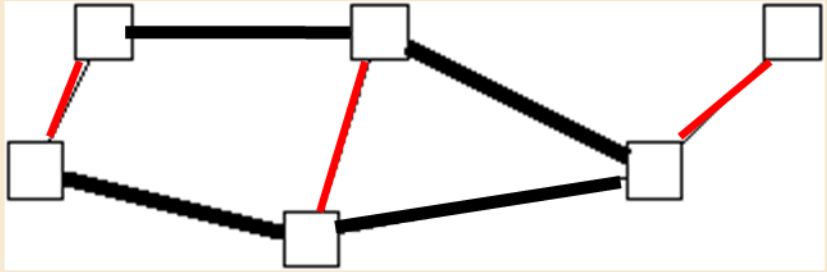
Couplage de poids max

Définition

- Un couplage dans un graphe simple est un ensemble M d'arêtes tel que M deux arêtes quelconques de M n'ont pas d'extrémité commune.
- Un couplage maximum est un couplage dont le nombre d'arêtes est maximal
- Un couplage parfait est un couplage qui est incident à tous les nœuds



Couplage de poids 2



Couplage max de poids 3

- Il est possible de trouver un couplage maximum en temps polynomial dans un graphe fini quelconque grâce à l'algorithme d'Edmonds

Couplage dans un graphe biparti

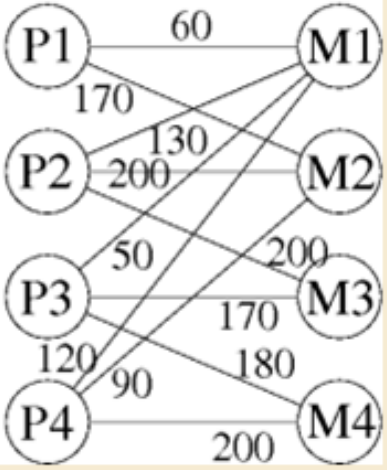
Définition

- La recherche d'un couplage maximum dans un graphe biparti est appelé le problème d'affectation.

Exemple

- Quatre produits P1, P2, P3 et P4 sont à affecter sur 4 machines dont les coûts d'exploitation dépendent du produit, comme indiqué sur le tableau. Sur quelle machine chaque produit doit-il être fabriqué pour minimiser le coût total ?

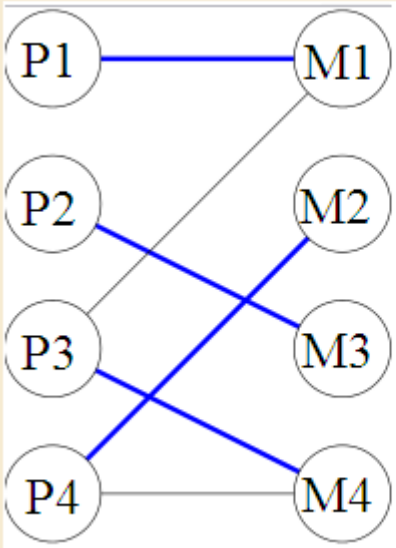
	M1	M2	M3	M4
P1	60	170	∞	∞
P2	130	200	200	∞
P3	50	∞	170	180
P4	120	90	∞	200



Couplage dans un graphe biparti

Exemple

- Le problème d'affectation peut être résolu par des algorithmes polynômiaux : comme l'algorithme hongrois.

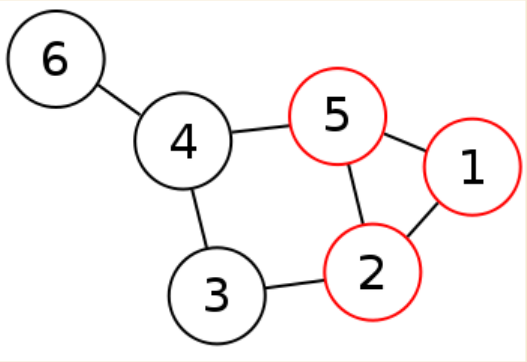


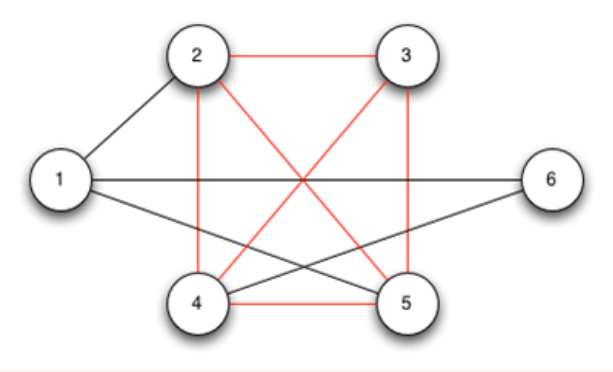
	M1	M2	M3	M4
P1	0	90	∞	∞
P2	20	70	0	∞
P3	0	∞	30	0
P4	50	0	∞	0

Clique de taille max

Définition

- Une clique d'un graphe non orienté est un sous-ensemble des sommets de ce graphe dont le sous-graphe induit est complet.





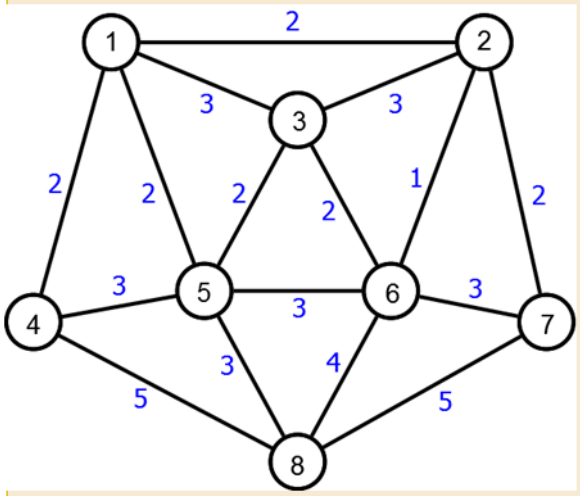
- La clique maximum d'un graphe est la clique dont le cardinal est le plus grand (c'est-à-dire qu'elle possède le plus grand nombre de sommets).
- Il s'agit d'établir si un graphe G donné contient une clique de cardinal au moins égal à un entier donné k. Lorsqu'on a constitué une liste de k sommets, il est trivial de vérifier s'ils forment une clique, et c'est pourquoi ce problème est de type NP.

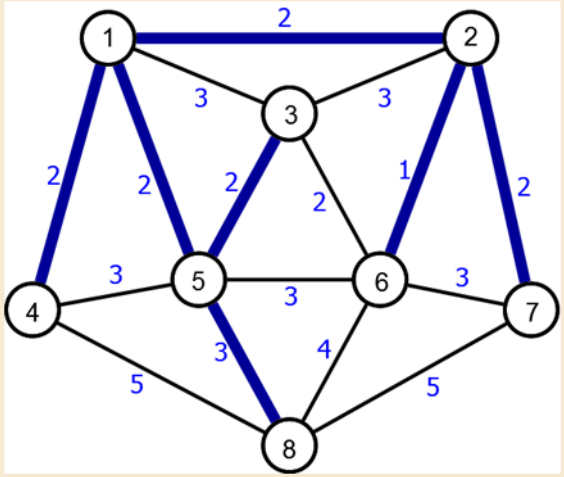
Arbre couvrant de poids min

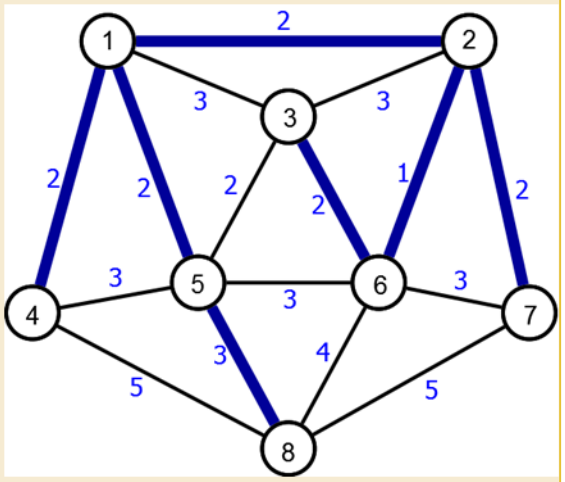
Définition

- Etant donné un graphe G non orienté, valué et connexe.
- Un arbre couvrant de G est un sous-graphe de G qui est un arbre et qui connecte tous les sommets de G.

- Un arbre couvrant de poids minimal est un arbre couvrant dont la somme des poids des arêtes est minimale.







Arbre couvrant de poids min

Résolution

- Il existe de nombreux algorithmes de recherche d'un arbre couvrant de poids minimal :
  - algorithme de Boruvka
  - algorithme de Prim
  - algorithme de Kruskal

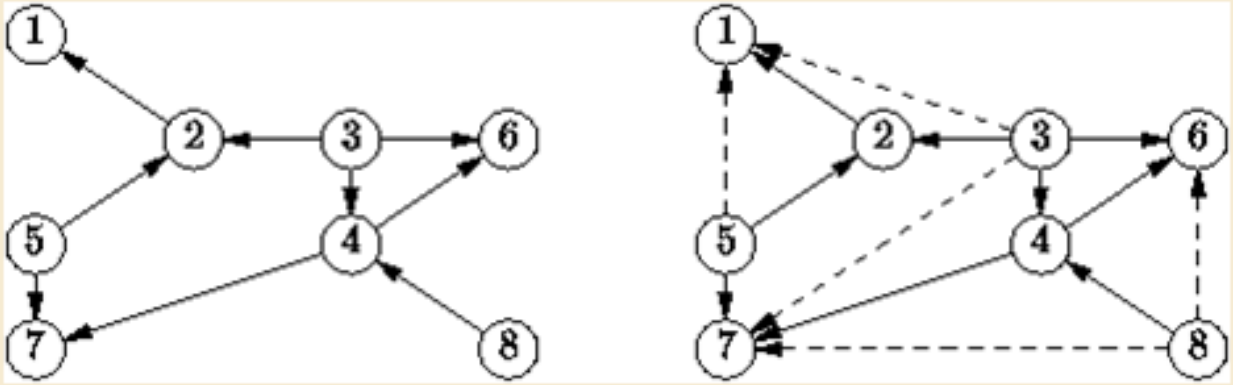
FERMETURE TRANSITIVE

Fermeture transitive

Définition

- La fermeture transitive d'un graphe (orienté ou non) **G = (V, E)** est définie par le graphe **G* = (V, E*)** où
  - **E*** = {(i, j) : il existe un chemin/chaîne reliant i à j dans G}

Exemple



Un graphe orienté **G** et sa fermeture transitive **G***

- L'arc (5,1) $\in$ **G*** : il existe un chemin de 5 à 1 = 5 - 2 - 1

Fermeture transitive

Intérêt

- Calculer la fermeture transitive :
  - d'un graphe non orienté sert à déterminer si ce graphe est connexe
  - d'un graphe orienté sert à déterminer si ce graphe est fortement connexe

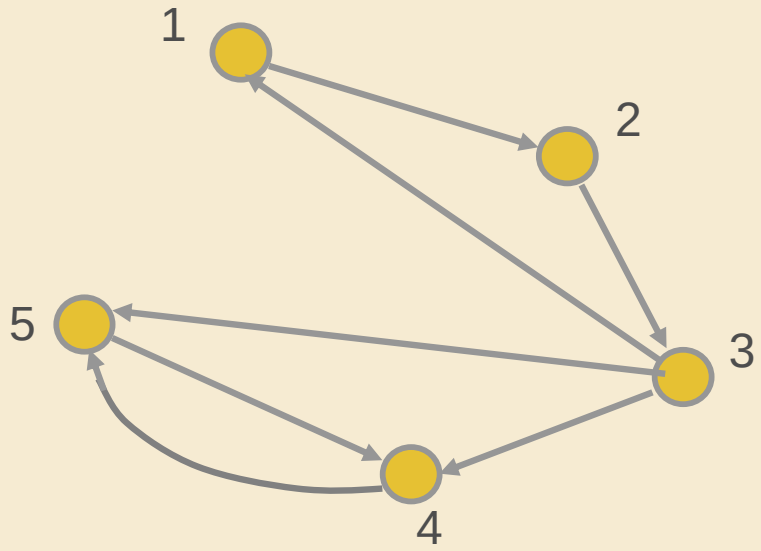
Remarques

- La fermeture transitive **G*** d'un graphe connexe (ou fortement connexe) **G** est un graphe complet
- Un graphe **G** est connexe (ou fortement connexe) si sa fermeture transitive **G*** est un graphe complet
- Deux sommets x, y sont de la même composante connexe de **G** si et seulement si il existe un arc (ou une arête) (x,y) dans la fermeture transitive **G*** de **G**.

Puissances successives

Exemple

- Dans la matrice $\mathbf{M^k}$, la terme $m_{ij} = 1$ ssi il existe un chemin (ou une chaîne) de longueur k allant du sommet i vers le sommet j
- Le calcul de la fermeture transitive d'un graphe peut se faire en additionnant les "puissances" successives de sa matrice d'adjacence



$$\boldsymbol{M} = \begin{pmatrix} \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{pmatrix}$$

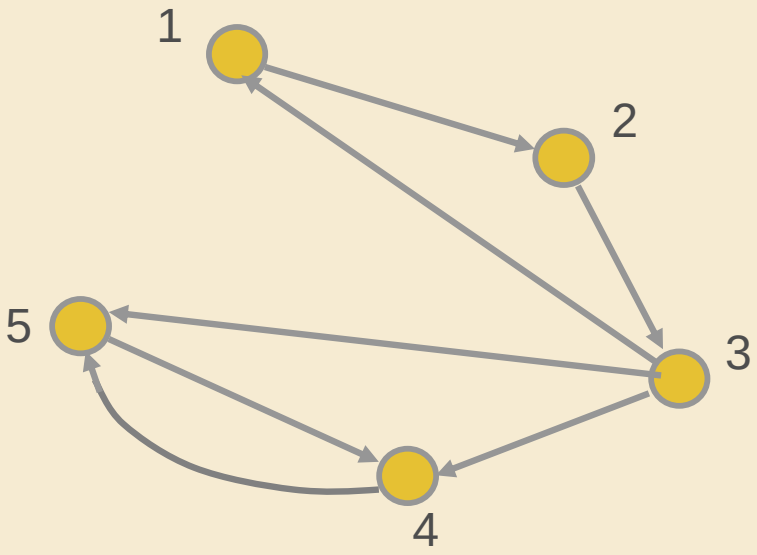
Un graphe orienté G et sa matrice d'adjacence

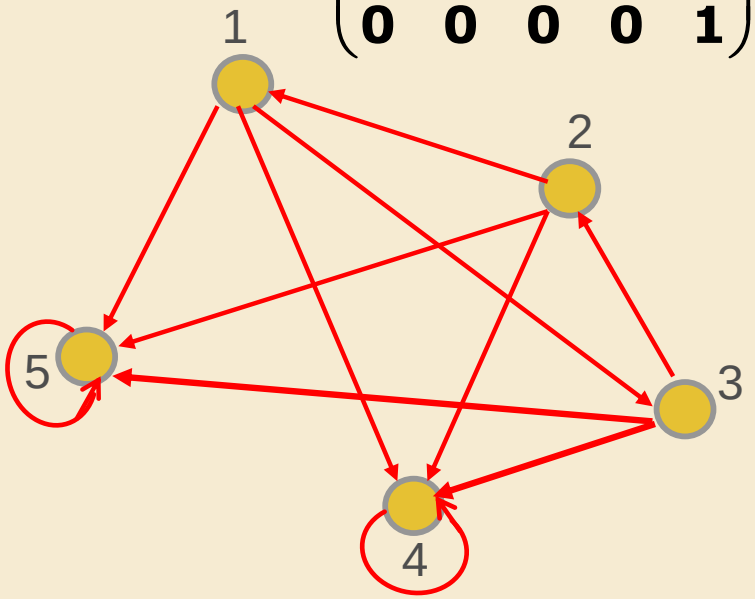
Puissances successives

Exemple

- Pour qu'il existe un chemin de longueur 2
- pour aller d'un sommet k à un sommet r,
- il faut qu'il existe un sommet i tel qu'il
- existe un chemin de longueur 1 de k
- vers i et un autre chemin de longueur 1
- de i vers r.

$$M^2 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$





Puissances successives

Exemple

- En additionnant **M et M^2**(+ et x booléenne), on obtient la matrice **$M + M^2$** des chemins de longueur inférieure ou égale à 2

$$M + M^2 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Puissances successives

Exemple

- En additionnant **M, M^2 et M^3**, on obtient la matrice **M + M^2 + M^3**,des chemins de longueur inférieure ou égale à 3

$$M + M^2 + M^3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Puissances successives

Exemple

- En additionnant **M, M^2 M^3 et M^4**, on obtient la matrice **$M+M^2+M^3+M^4$**,des chemins de longueur inférieure ou égale à 4

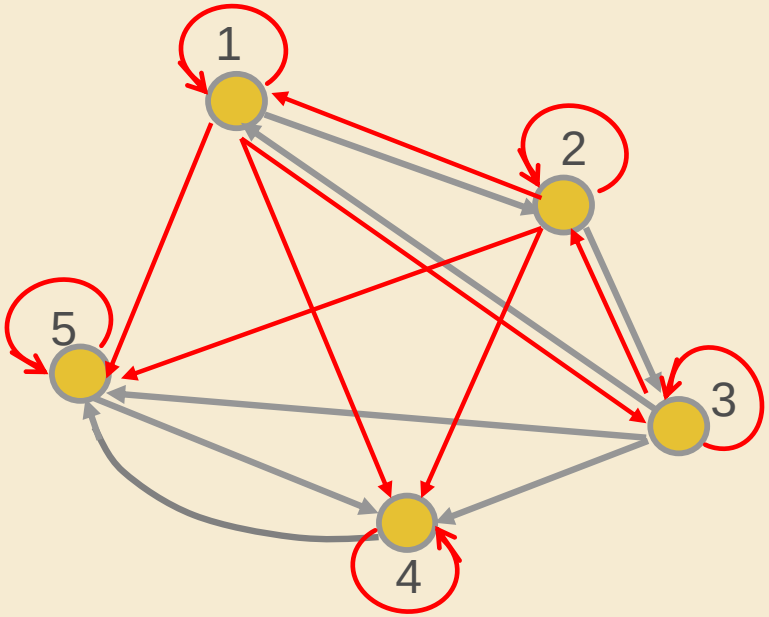
$$\sum_{i=1}^{4} M^i = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Puissances successives

Exemple

- On obtient $\mathbf{M+M^2+M^3+M^4 = M+M^2+M^3}$
- donc $\mathbf{M+M^2+M^3}$ est la matrice d'adjacence de la fermeture transitive du graphe **G**

$$\sum_{i=1}^{3} M^i = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$



- La complexité de cet algorithme est $o(n^4)$

Algorithme de Roy-Warshall

Principe de l'algorithme

- Cet algorithme ajoute des arcs au graphe G jusqu'à obtenir la fermeture transitive

- **procédure Roy-Warshall** (G un graphe)
  - **pour tout** sommet y de G **faire**
    - **pour tout** arc (x,y) de G entrant dans y, où x≠y, faire
      - **pour tout** arc (y,z) de G sortant de y, où z≠y, faire
        - si (x,z) n'est pas un arc de G alors ajouter l'arc (x,z)
        - fin si
      - **fin pour (y,z)**
    - **fin pour (x,y)**
  - **Fin pour y**
- Fin procédure

- **La complexité** de cet algorithme est : **$o(n^3)$**

Algorithme de Roy-Warshall

Principe de l'algorithme

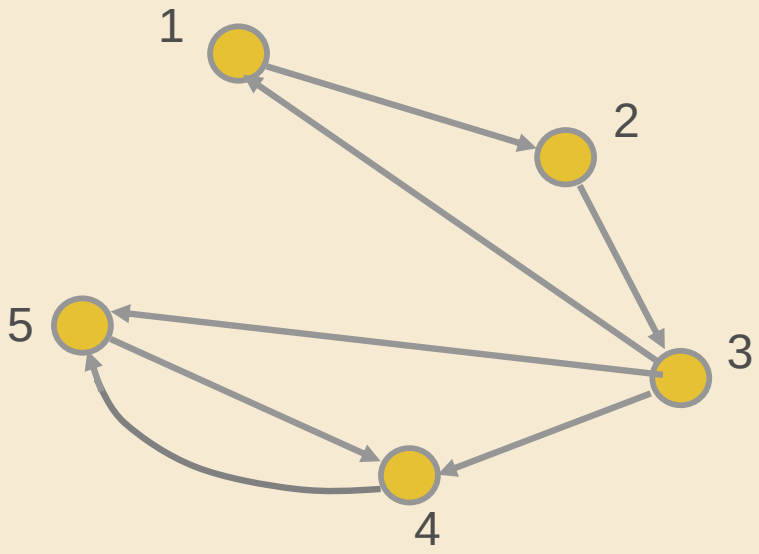
- Cet algorithme utilise la matrice d'adjacence pour calculer la matrice d'adjacence de la fermeture transitive d'un graphe simple

- **procédure Roy-Warshall(M** matrice de booléen**) ;**
  - A ← M (A = matrice de la fermeture transitive)
  - **pour** i = 1 à n faire
  - A[i, i] = 1
  - **finpour**
  - **pour** k = 1 à n faire
  - **pour** i := 1 à n faire
  - **pour** j := 1 à n faire
  - A[i, j] := A[i, j] ou (A[i, k] et A[k, j]) ;
  - **finpour**
  - **finpour**
  - **finpour**
- Fin procédure

- La **complexité** de cet algorithme est : **o(n^3)**

Algorithme de Roy-Warshall

Exemple



$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Un graphe orienté G et sa matrice d'adjacence

Algorithme de Roy-Warshall

Exemple

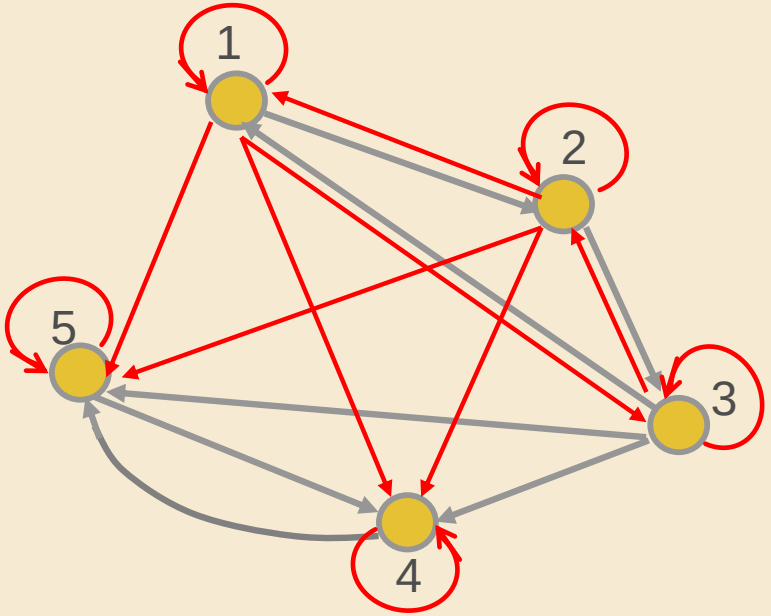
$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad A_0 = \begin{pmatrix} \underline{1} & 1 & 0 & 0 & 0 \\ 0 & \underline{1} & 1 & 0 & 0 \\ 1 & 0 & \underline{1} & 1 & 1 \\ 0 & 0 & 0 & \underline{1} & 1 \\ 0 & 0 & 0 & 1 & \underline{1} \end{pmatrix} \quad A_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & \underline{1} & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 1 & 1 & \underline{1} & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \quad A_3 = \begin{pmatrix} 1 & 1 & 1 & \underline{1} & \underline{1} \\ \underline{1} & 1 & 1 & \underline{1} & \underline{1} \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \quad A_4 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Algorithme de Roy-Warshall

Exemple

$$A_4 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$



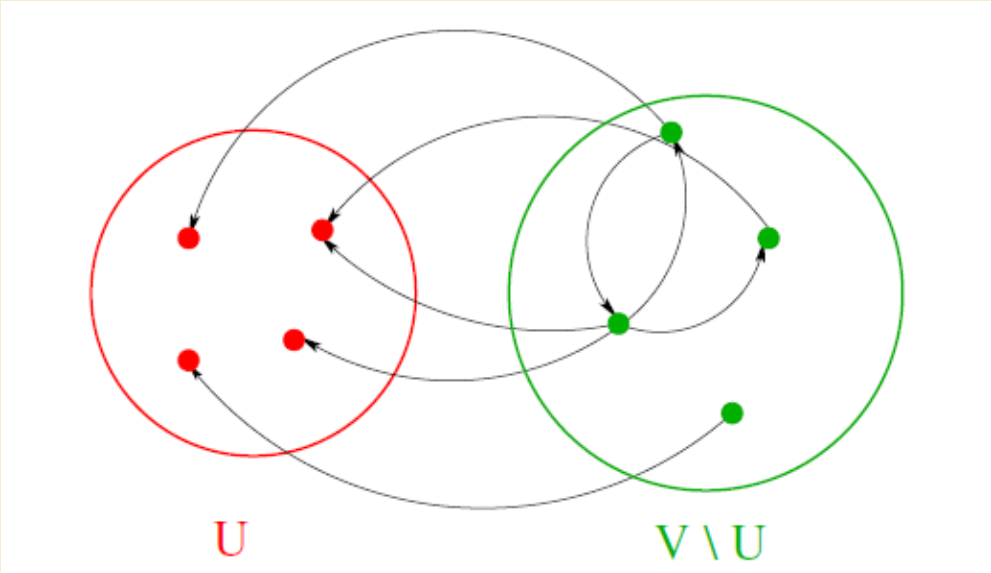
G admet deux composantes fortement connexes :
F = {1, 2, 3} et H = {4, 5}

NOYAU D'UN GRAPHE

Noyau d'un graphe

Définition

- Soit **G =(S, A)** un graphe orienté. Le noyau **U** de **G** est formé d'une partie des sommets de **G**, telle que
- les sommets de **U** n'ont aucun arc les joignant deux à deux
- tout sommet en dehors de **U** admet au moins un successeur dans **U**



ROUGE sommets du noyau **VERT** sommets à l'extérieur du noyau

Noyau d'un graphe

Intérêt

- Le noyau d'un graphe est très utile pour déterminer la solution d'un type particulier de jeux, il est utile dans la cryptographie

Complexité

- Chvàtal a montré en 1973 que trouver un noyau dans un graphe était un problème NP-complet.

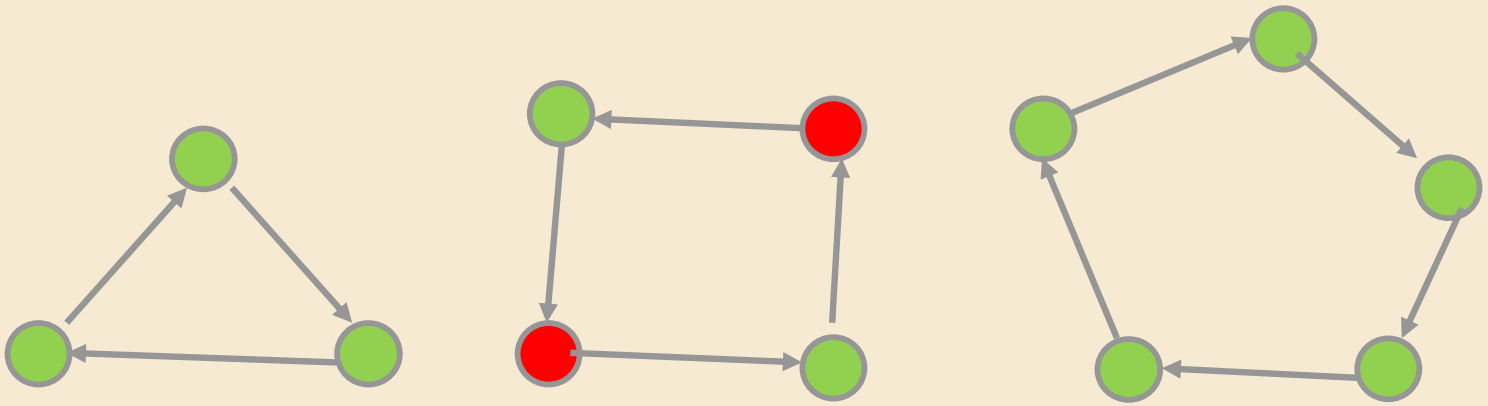
Exemple : jeu à deux joueurs

- Jeux finis à deux joueurs sans information cachée avec toujours un vainqueur. Un des deux joueurs possède une stratégie gagnante (exemple jeu des allumettes)
- Graphe orienté
  - Sommets : configurations du jeu
  - Arcs : déplacements
- Noyau = Ensemble des positions gagnantes
  - Celui qui possède une stratégie gagnante se déplace toujours vers un sommet du noyau

Noyau d'un graphe

Exemples de graphes en forme de circuit

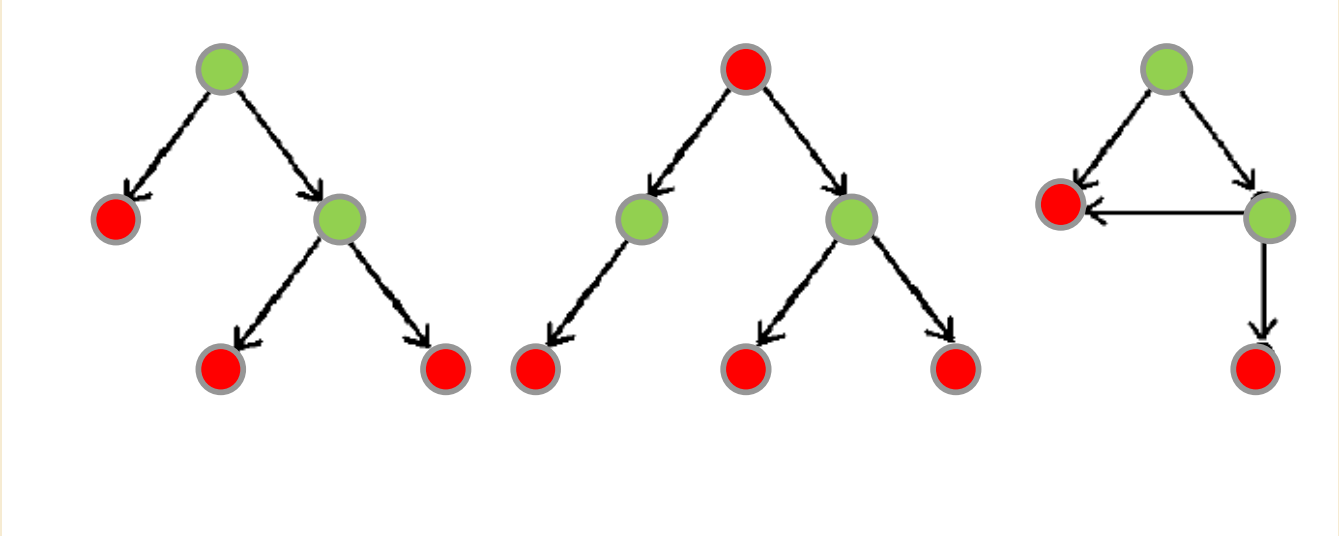
- Deux cas pour les circuits :
  - ceux qui ont un nombre impair de sommets n'ont pas de noyau
  - ceux qui ont un nombre n pair de sommets en ont un noyau et |U|= n/2



Noyau d'un graphe

Exemples graphes sans circuit

- Il s'agit là de graphes sans circuit possédant un noyau unique.

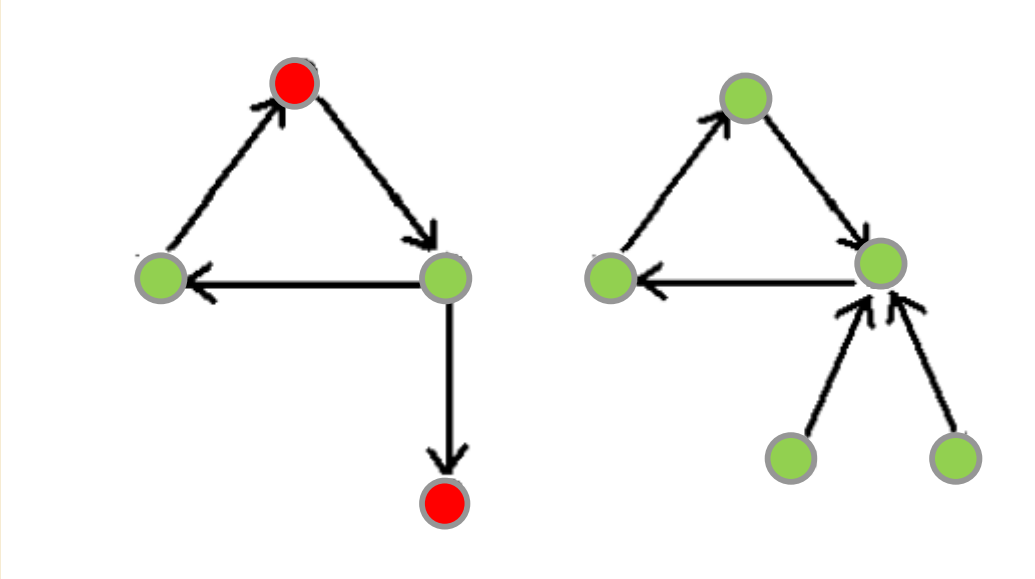


- Remarquons que les sommets sans successeur sont forcément des éléments du noyau, et que leurs prédécesseurs ne sont pas dans le noyau.

Noyau d'un graphe

Exemples graphes avec circuit

- Il s'agit là de graphes avec circuit



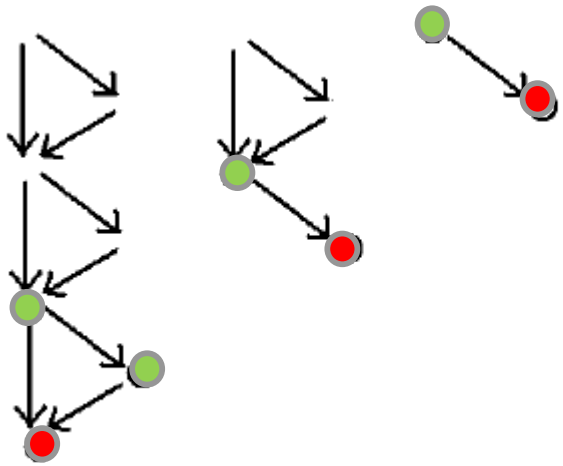
- Ces deux graphes ont un cycle. Le premier possède un noyau, le deuxième n'en a pas.

Noyau d'un graphe

Théorème

- Si un graphe orienté n'a pas de circuit de longueur impaire, il admet un noyau
- Si un graphe orienté ne possède aucun circuit, il admet un noyau et celui-ci est unique.

Algorithme de construction d'un noyau



Construction du noyau :
1) on met le sommet qui n'a pas de successeurs dans le noyau et ses deux prédécesseurs hors du noyau.
2) on prend la partie restante du graphe, en mettant le sommet d'en bas dans le noyau, et son prédécesseur en dehors.
3) on prend ce qui reste du graphe, en mettant le sommet d'en bas dans le noyau, et son prédécesseur en dehors.

CHEMINS OPTIMAUX DANS UN GRAPHE

Problèmes de chemins optimaux

Définition

- Soit **G =(S, A, W)** un graphe valué orienté. Le **coût d'un chemin p =<s_0, s_1, s_2, …, s_k>** est égal à la somme des coûts des arcs composant le chemin, c'est à dire,

$$cout(p) = \sum_{i=1}^{k} cout(s_{i-1}, s_i)$$

- Le **coût d'un chemin** sera aussi appelé **poids du chemin**.

- Le coût (ou poids) d'un **plus court chemin** entre deux sommets **s_i et s_j** est noté **$\delta(s_i , s_j)$** et est défini par

  - **$\delta(s_i , s_j) = +\infty$** si il n'existe pas de chemin entre s_i et s_j

  - **$\delta(s_i , s_j) = min\{cout(p) / p =$ chemin de s_i à $s_j\}$** si il existe un chemin entre s_i et s_j

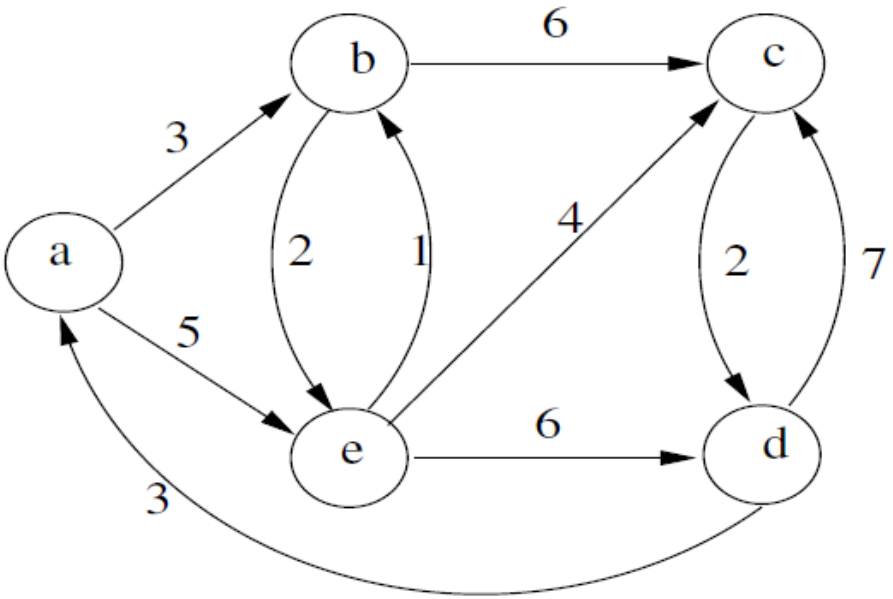
Types de problèmes

Types de problèmes

- **3 types de problèmes :**
  - **Problème A** : trouver un plus court chemin du sommet s au sommet t
  - **Problème B** : étant donné un sommet de départ s, trouver un plus court chemin de s vers tout les autres chemins
  - **Problème C** : trouver un plus court chemin entre tout couples de sommets c-à-d calculer une matrice nxn (distancier)
- Un algorithme pour un de ces problèmes peut être utilisé pour résoudre les deux autres.
- La grande majorité des problèmes concerne le **problème B**

Problèmes de chemins optimaux

Exemple



- Nous avons : δ**(a; b) = 3,** δ**(a; e) = 5,** δ**(a; c) = 9,** δ**(a; d) = 11**

Existence d'un chemin optimal

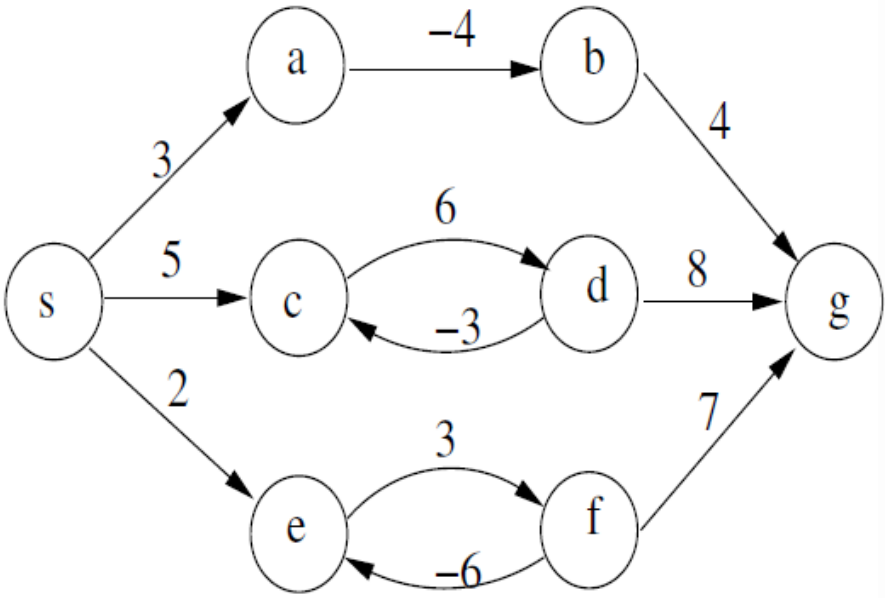
Conditions d'existence

- Conditions d'existence d'un plus court chemin :
  - s'il existe un chemin entre deux sommets u et v contenant un **circuit de coût négatif**, alors :
    - δ**(u; v) = -**∞,
    - il n'existe pas de plus court chemin entre u et v.

- Un circuit négatif est appelé un circuit absorbant

Problèmes de chemins optimaux

Définition



Le chemin < s; e; f; e; f; g > contient le circuit < e; f; e > de coût négatif -3.

Autrement dit, à chaque fois que l'on passe dans ce circuit, on diminue de 3 le coût total du chemin.

Par conséquent, $\delta(s; g) = -\infty$ et il n'existe pas de plus court chemin entre s et g.

Algorithme de Dijkstra

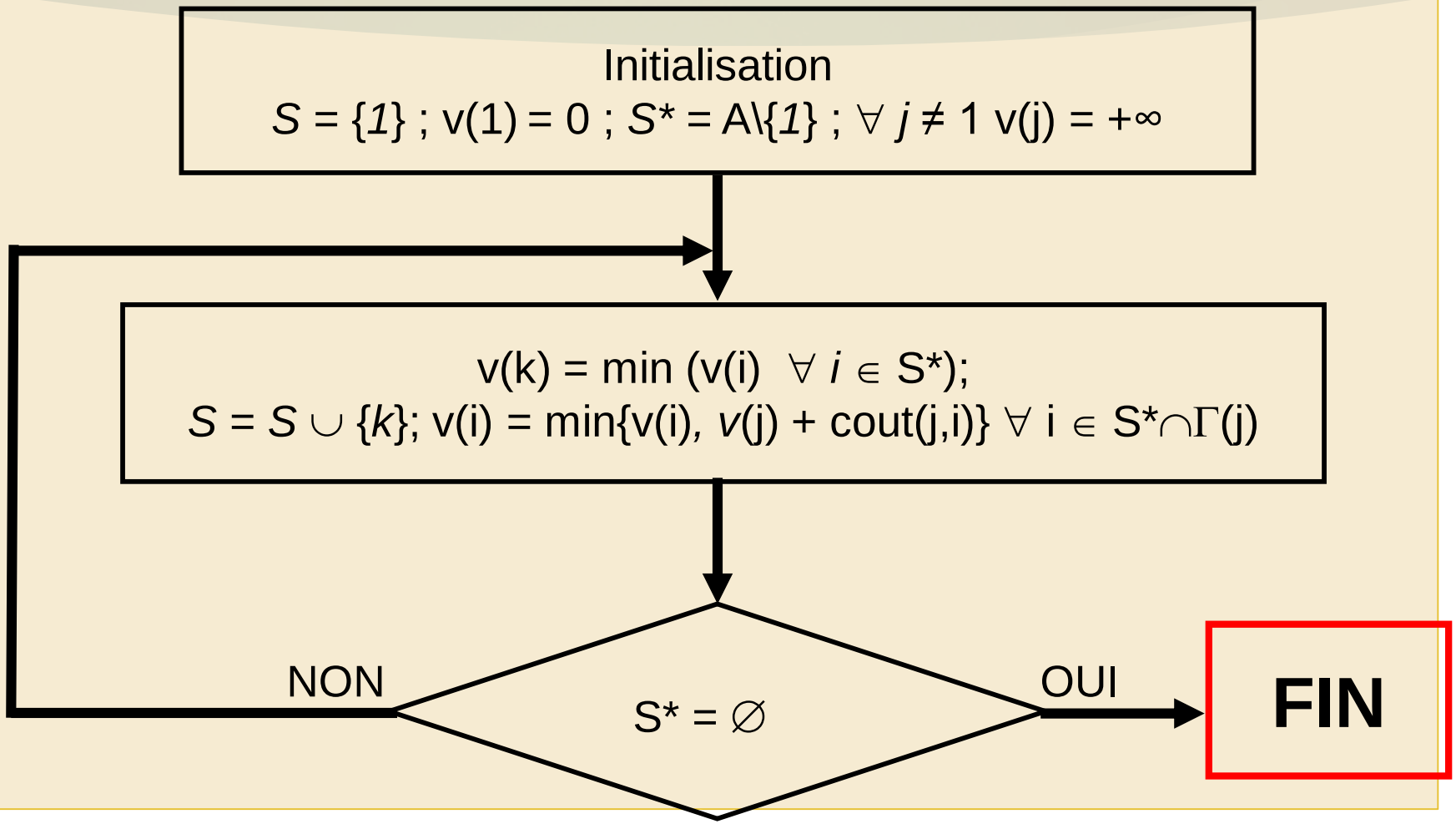
Validité

- **Validité** : couts des arcs non négatifs
- Le problème de la non existence d'un circuit négatif est donc évité
- **Principe :**
  - A chaque itération, un sommet x reçoit son étiquette définitive v(x) (on dit qu'il est fixé)
  - Un tableau de booléens indique les sommets fixés

- Etiquettes initiales à $+\infty$, sauf v(s) = 0
- **Itérations principales :**
  - Calcul du sommet x d'étiquette minimale
  - Pour tout successeur y de x, on regarde si le chemin passant par x améliore le chemin déjà trouvé de s à y. On remplace donc v(y) par Min(v(y), v(x) + cout(x,y))
- En cas d'amélioration, on mémorise le chemin : P(y) = x
- Si tout les sommets sont descendants de s, il y a n itérations. Si non il faut stopper l'algorithme quand v(x) est infini.

Algorithme de Dijkstra

Algorithme



Algorithme de Dijkstra

Algorithme

- **(a) Initialisations**
  - S* = {2, 3, …,n}
  - v(1) = 0
  - v(i) = cout(1,i) si $i \in \Gamma(1)$
  - v(i) = $+\infty$ sinon
- **(b) Sélection**
  - Sélectionner j tel que v(j) = min (v(k)) pour tout $k \in S^*$
  - $S^* \leftarrow S^* \backslash \{j\}$
  - Si $S^* = \varnothing$ alors FIN
- **(c) Mise à jour**
  - Faire pour tout $i \in S^* \cap \Gamma(j)$ v(i) = min(v(i),v(j) +cout(j,i))
  - Retourner en (b)

Exemple

Exemple

(a) S = {2, 3, 4, 5, 6}

v(1) = 0, v(2) = 7, v(3) = 1, v(4) = v(5) = v(6) = +∞

(b) j = 3, S* = {2, 4, 5, 6},

(c) $S^* \cap \Gamma(3) = \{2, 5, 6\}$

v(2) = min(7,1+5) = 6, v(5) = min(1,1+2) = 3,

v(6) = min(1,1+7) = 8

(b) j = 5, S* = {2, 4, 6}

(c) $S^* \cap \Gamma(5) = \{2, 4\}$

v(2) = min(6,3+2) = 5,

v(5) = min(1,3+5) = 8

(b) j = 2, S* = {4, 6}

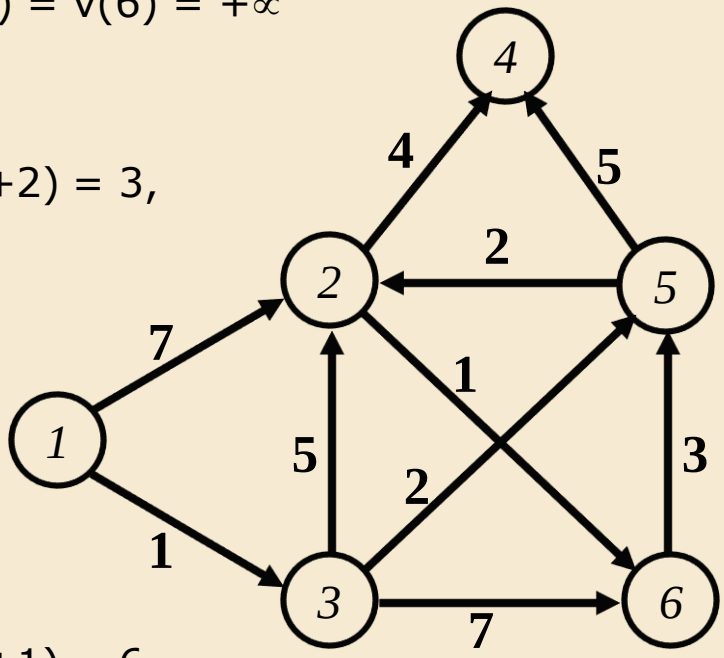
(c) $S^* \cap \Gamma(2) = \{4, 6\}$

v(4) = min(8,5+4) = 8, v(6) = min(8,5+1) = 6

(b) j = 6, S* = {4}

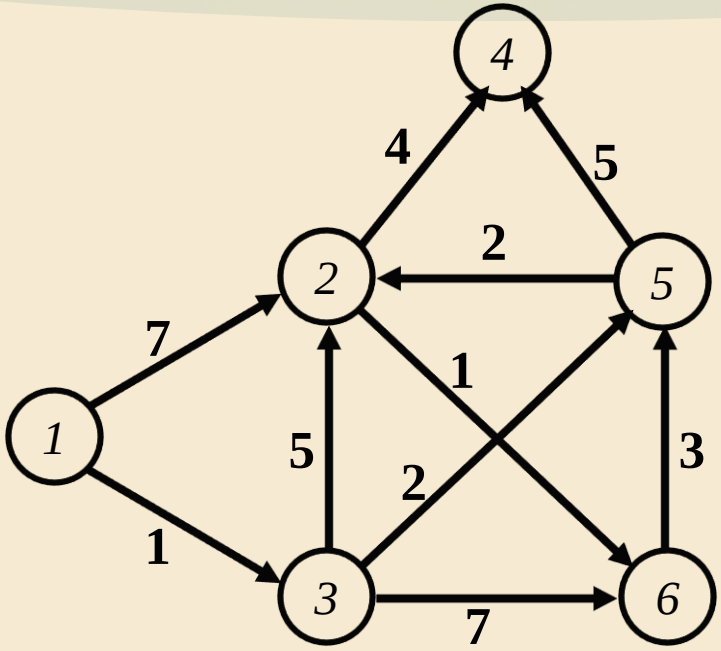
(c) $S^* \cap \Gamma(2) = \varnothing$

(b) j = 4, S* = ∅



Exemple

Exemple



Les longueurs des plus courts chemins du sommet 1 vers les autres sommets sont :
v(1) = 0, v(2) = 5, v(3) = 1, v(4) = 8, v(5) = 3 et v(6) = 6

Algorithme de Bellman

Validité & Principe

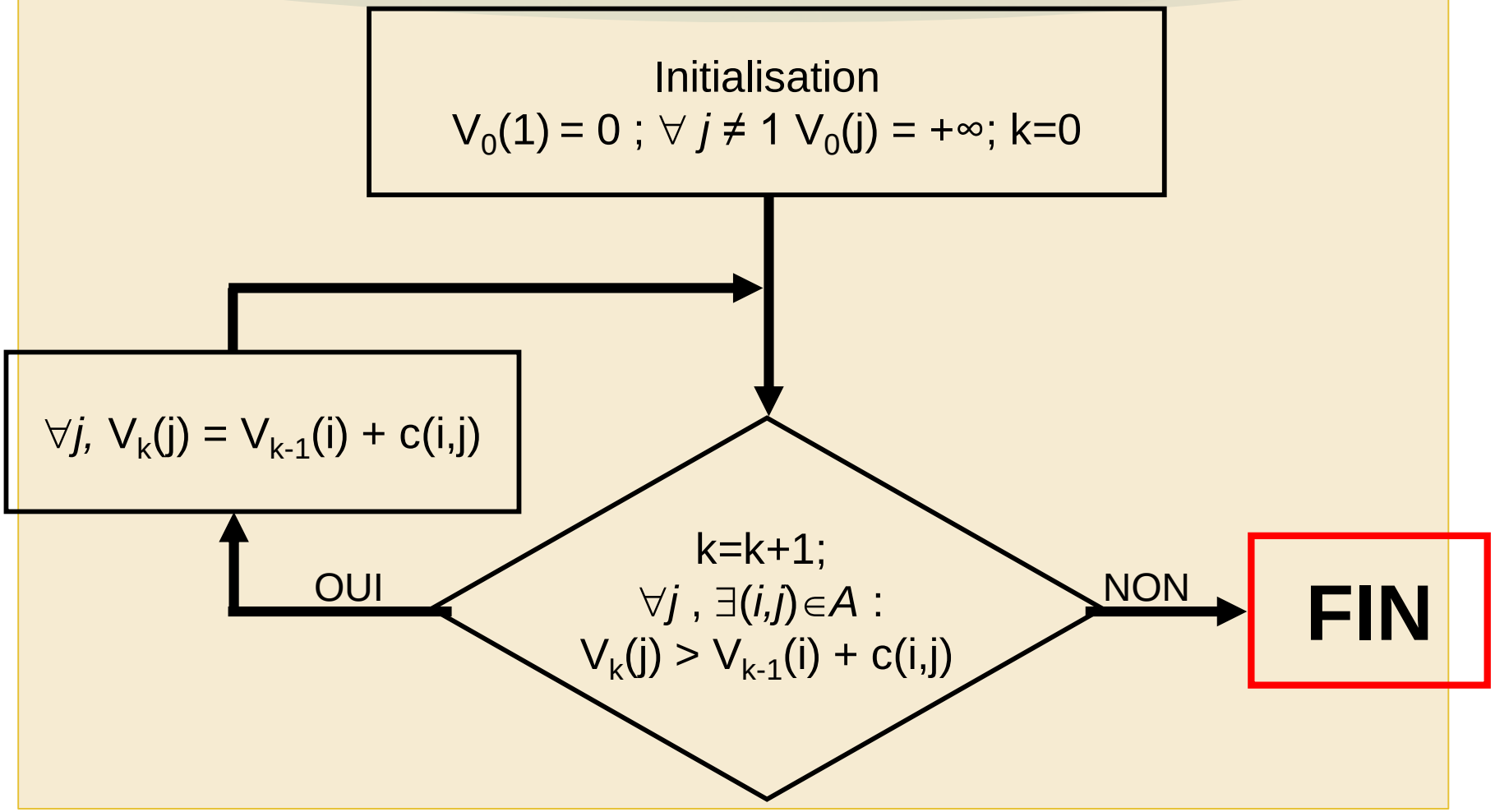
- **Validité** : couts des arcs positifs ou négatifs
- Algorithme prévu pour des valuations quelconques (coûts positifs ou négatifs)
- Algorithme qui peut détecter la présence d'un circuit absorbant
- **Principe :**
  - Un chemin optimal de longueur k de s à y s'obtient à partir des chemins optimaux de longueur k-1 de s vers tout prédécesseurs x de y
  - $V_k(j)$ = la valeur des plus courts chemins d'au plus k arcs du sommet 1 au sommet j

- Algorithme se base sur la relation de récurrence suivante :
  - $V_0(1) = 0$
  - $V_0(j) = +\infty$, pour tout $j \neq 1$
  - $V_k(j) = \min \{V_{k-1}(i) + c(i,j) / j \in \Gamma(i)\}$, $k > 0$

  - A chaque itération, les étiquettes sont corrigées

Algorithme de Bellman

Algorithme



Algorithme de Bellman

Algorithme

- **(a) Initialisations**
  - $V_0(1) = 0$;
  - $\forall\, j \neq 1\ V_0(j) = +\infty$
  - $k = 0$

- **(b) Mise à jour**
  - $k = k + 1$
  - Pour j allant de 1 à |V|
    - Pour tout i prédécesseur de j
      - Si $V_{k-1}(i) \neq \infty$ *et* $V_k(j) > V_{k-1}(i) + c(i,j)$ alors
        - $V_k(j) = V_{k-1}(i) + c(i,j)$
  - Si V_{k-1} est stable ou $k = |V|$ alors FIN
  - Sinon Retourner en (b)

Exemple

Exemple

(a) k=0

$v_0(1) = 0$, $v_0(2) = v_0(3) = v_0(4) = v_0(5) = v_0(6) = +\infty$

(b) k = 1

$v_1(2) = \min(+\infty,7) = 7$, $v_1(3) = \min(+\infty,1) = 1$,

$v_1(4) = v_1(5) = v_1(6) = +\infty$

(b) k = 2

$v_2(2) = \min(7,5+1) = 6$, $v_2(4) = \min(+\infty,6+4) = 10$

$v_2(5) = \min(+\infty,1+2) = 3$,

$v_2(6) = \min(+\infty,7) = 7$

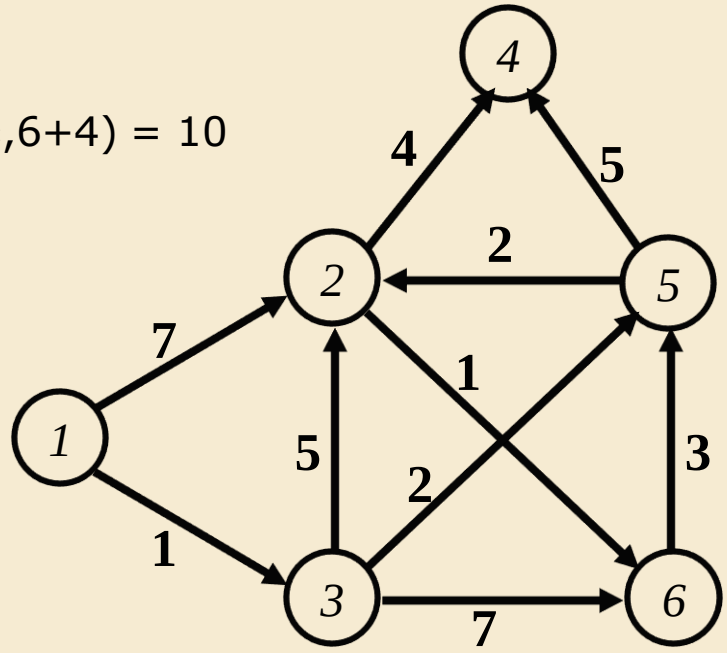
(b) k = 3

$v_3(4) = \min(10,3+5) = 8$

(b) k = 4

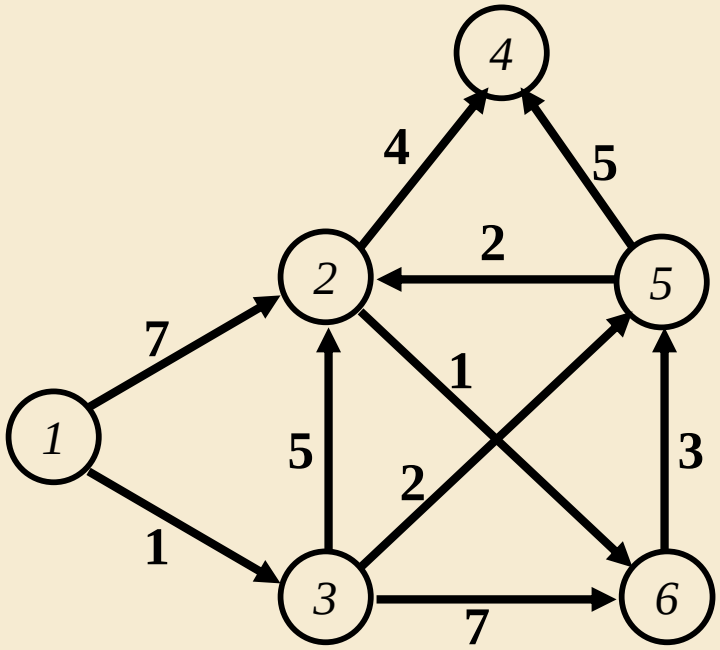
aucune amélioration, v_3 est stable

Fin



Exemple

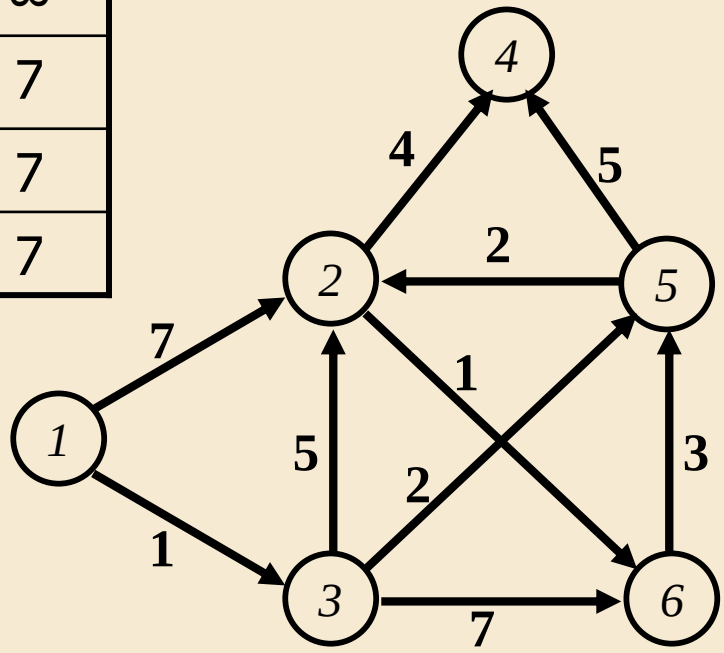
Exemple



Exemple

Exemple

k	1	2	3	4	5	6
0	0	∞	∞	∞	∞	∞
1	0	7	1	∞	∞	∞
2	0	6	1	10	3	7
3	0	5	1	8	3	7
4	0	5	1	8	3	7



PROBLEMES DE FLOTS DANS UN GRAPHE

Problèmes de flots

Introduction

- Un flot dans un graphe représente la circulation d'un produit sur les arcs de ce graphe.

- Les problèmes de flots consistent à :
  - maximiser le flots d'un produit entre deux sommets donnés : **problèmes de flot maximal**
  - minimiser le coût de transport d'une certaine quantité de ce produit : **problèmes de flot de coût minimal**

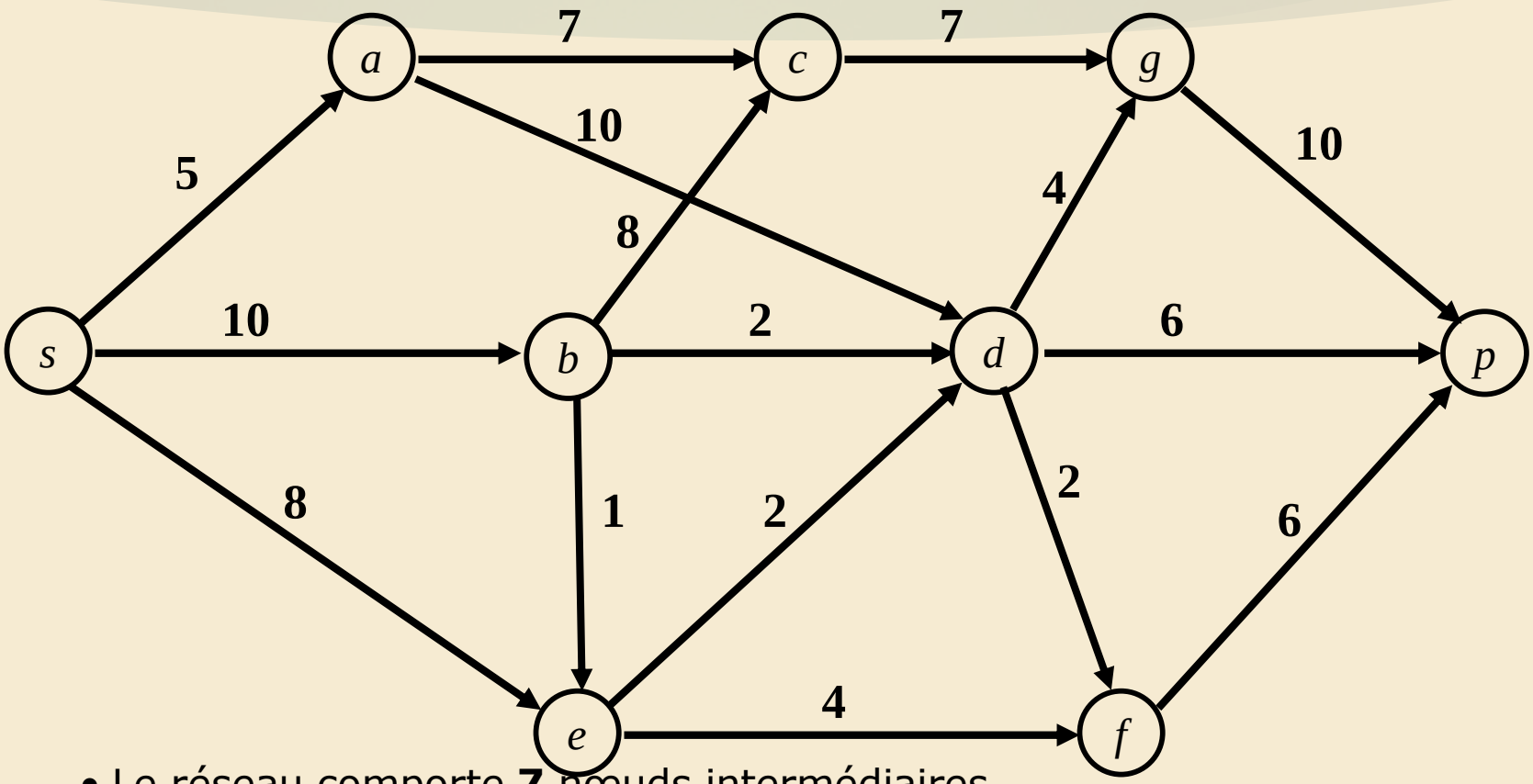
- Nous traitons ici le problème de flot maximal

Problèmes de flots

Introduction

- **Nous nous intéressons aux graphes (réseaux de transport) suivants :**
  - **connexe sans boucle** et **asymétrique** :
    - $(x, y) \in A \Rightarrow (y, x) \notin A$
  - où chaque arc (x,y) est valué par un nombre
    - **C(x,y)** ≥ **0**, appelé **capacité** de l'arc (x,y)
  - admet une entrée appelée **source** (notée s) : sommet n'ayant pas de prédécesseur
  - et une sortie appelée **puits** (notée p) : sommet n'ayant pas de successeur

Problèmes de flots



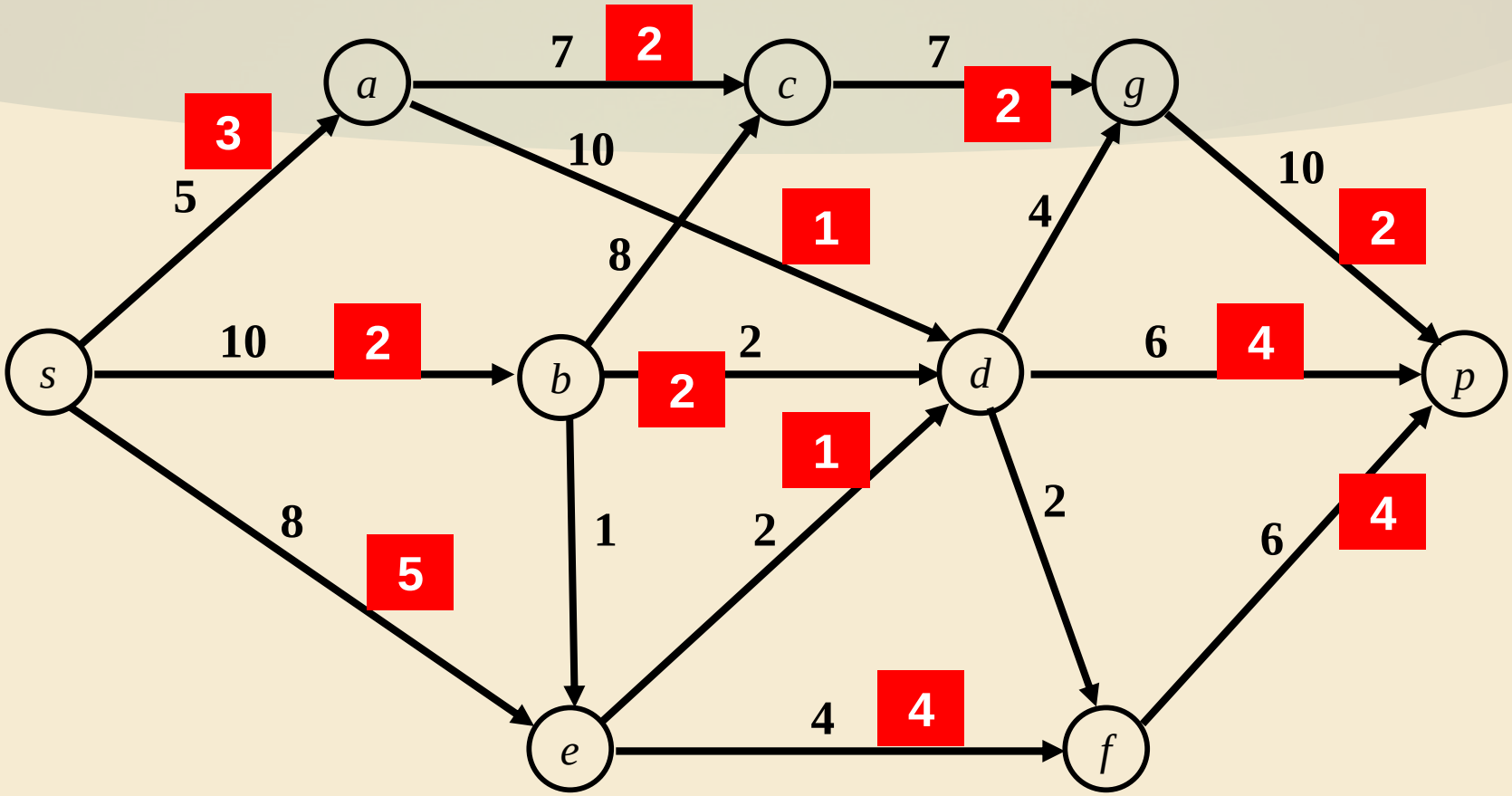
- Le réseau comporte **7** nœuds intermédiaires.
- La capacité de l'arc ***(c,g)*** est de **7**, celle de l'arc ***(a,d)*** est de **10**

Problèmes de flots

Définition

- Un **flot *F*** sur un réseau ***N=(V,A)*** est une valuation positive des arcs, c'est à dire une application de ***A*** dans ***R+***, qui vérifie les deux propriétés suivantes :
  - Pour tout arc (x,y)$\in$***A***, $\mathbf{0} \leq \boldsymbol{F(x,y)} \leq \boldsymbol{c(x,y)}$
  - Pour tout sommet intermédiaire $\boldsymbol{x} \in \boldsymbol{V} \setminus \{\boldsymbol{s,p}\}$
    - $\Sigma_y \boldsymbol{F(y,x)} = \Sigma_y \boldsymbol{F(x,y)}$
- La propriété 2 (règle de cohérence) $\Leftrightarrow$ la quantité totale du flot qui arrive au sommet x (autre que s et p) doit être égale à la quantité qui sort du sommet x

Problèmes de flots



- Le flot entrant en **b** a une valeur de **2**.
- La *valeur* du flot est définie comme le flux net sortant de ***s*** ou entrant dans ***p***. Sur cet exemple le flot a une valeur **3+2+5 = 10**

Problèmes de flots

Définition du flot maximal

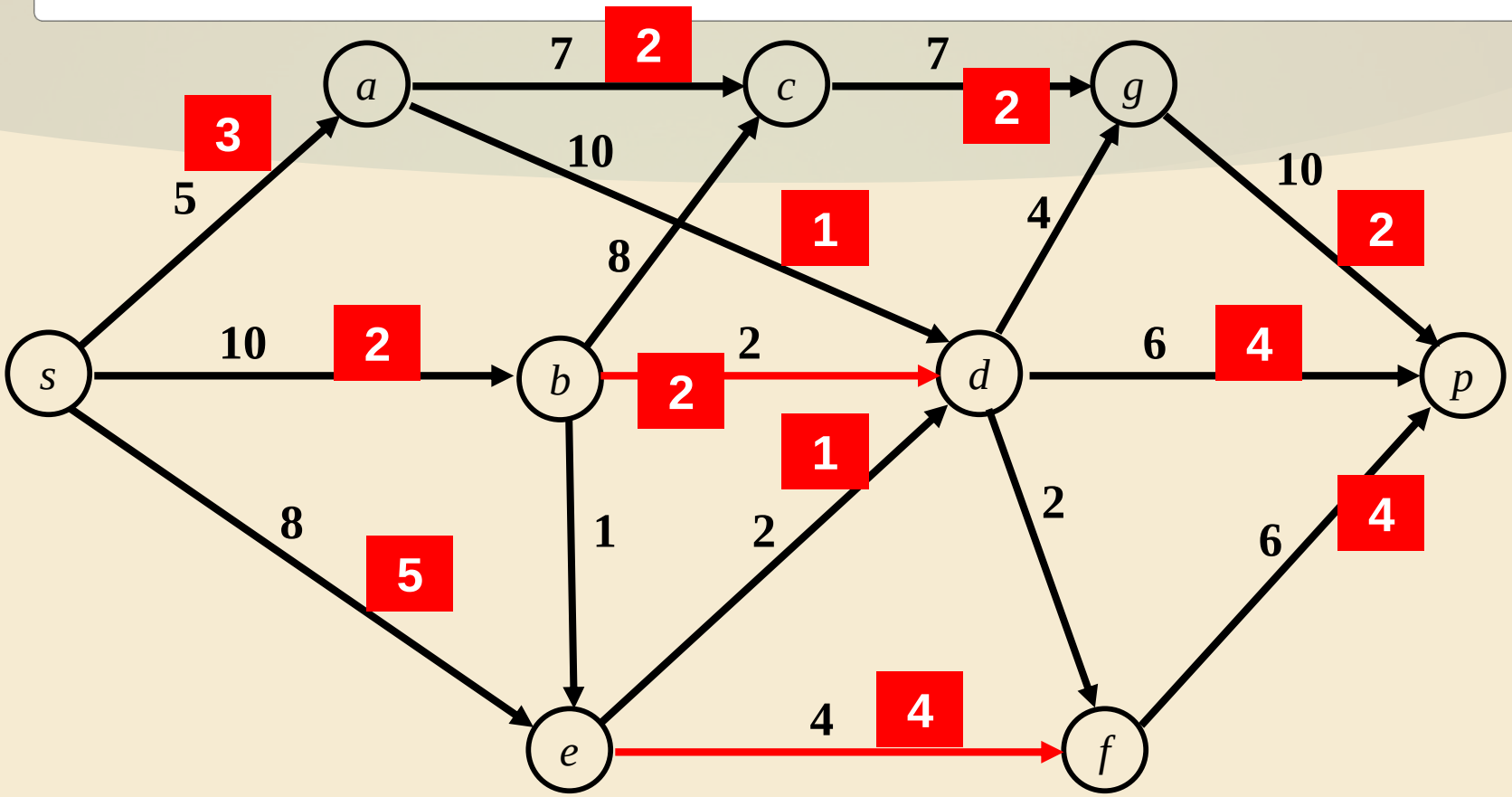
- La somme $F^-(x) = F(y,x)$ est le *flot entrant* au sommet x
- La somme $F^+(x) = F(x,y)$ est le *flot sortant* du sommet x
- La **valeur** $|F|$ d'un flot F est définie comme le flot sortant moins le flot entrant en s :
  - $|F| = F^+(s) - F^-(s)$
- Le problème du **Flot Maximal** consiste à trouver un flot F_{max} de valeur maximale sur le réseau N.
- c-à-d un flot qui permet d'acheminer la plus grande quantité possible de la source au puits

Problèmes de flots

Définitions

- Un arc *(x,y)* est **saturé** par un flot ***F*** si la valeur du flot sur l'arc est égale à sa capacité :
  - ***F(x,y) = c(x,y)***
- Un **chemin est saturé** si l'un de ses arcs est saturé.
- La **capacité résiduelle** d'un arc ***(x,y)*** est la quantité ***c(x,y) - F(x,y)*** de flot pouvant encore transiter par lui.
- La **capacité résiduelle** d'un chemin est la plus petite capacité résiduelle de ses arcs.
- **Saturer** un chemin ***T*** de ***s*** à ***p*** consiste à augmenter le flot de ses arcs de la capacité résiduelle du chemin.

Problèmes de flots



- La capacité résiduelle de l'arc ***(a,c)*** est **7 - 2 = 5**
- L'arc ***(e,f)*** a une capacité résiduelle nulle : il est saturé.
- Le chemin **(s,a,c,g,p)** n'est pas saturé
- Le chemin **(s,b,d,p)** est saturé

Algorithme de Ford-Fulkerson

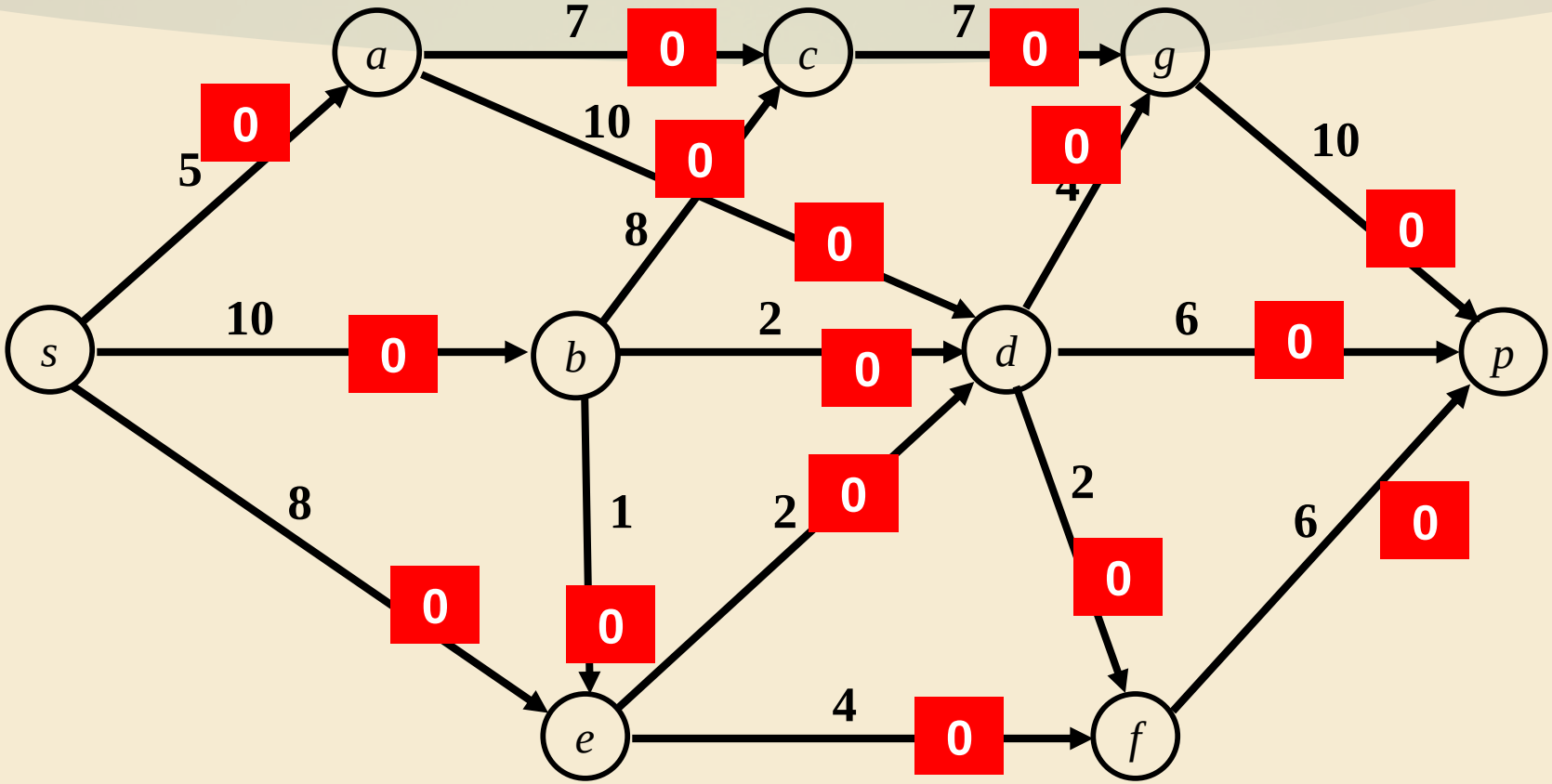
Algorithme

- Un **Flot est complet** si tout chemin allant de **s** à **p** sont saturés

- **Principe de l'algorithme :**
- On part d'un flot complet
- et on l'améliore progressivement

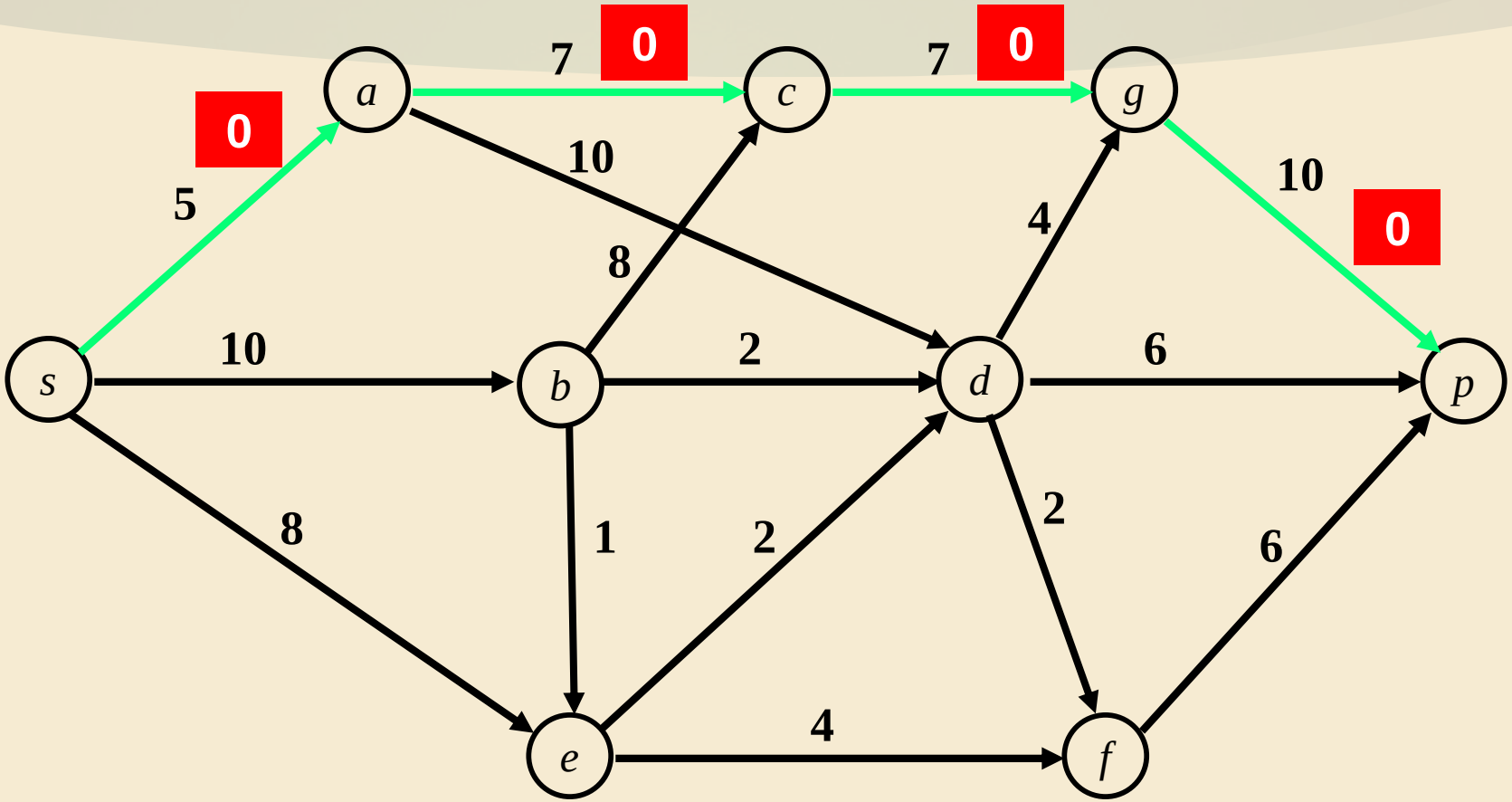
- **Procédure de Ford et Fulkerson :**
- Permet de trouver une chaîne améliorante
- S'il n'en existe pas le flot est maximum

Algorithme de Ford-Fulkerson



Recherche d'un flot complet

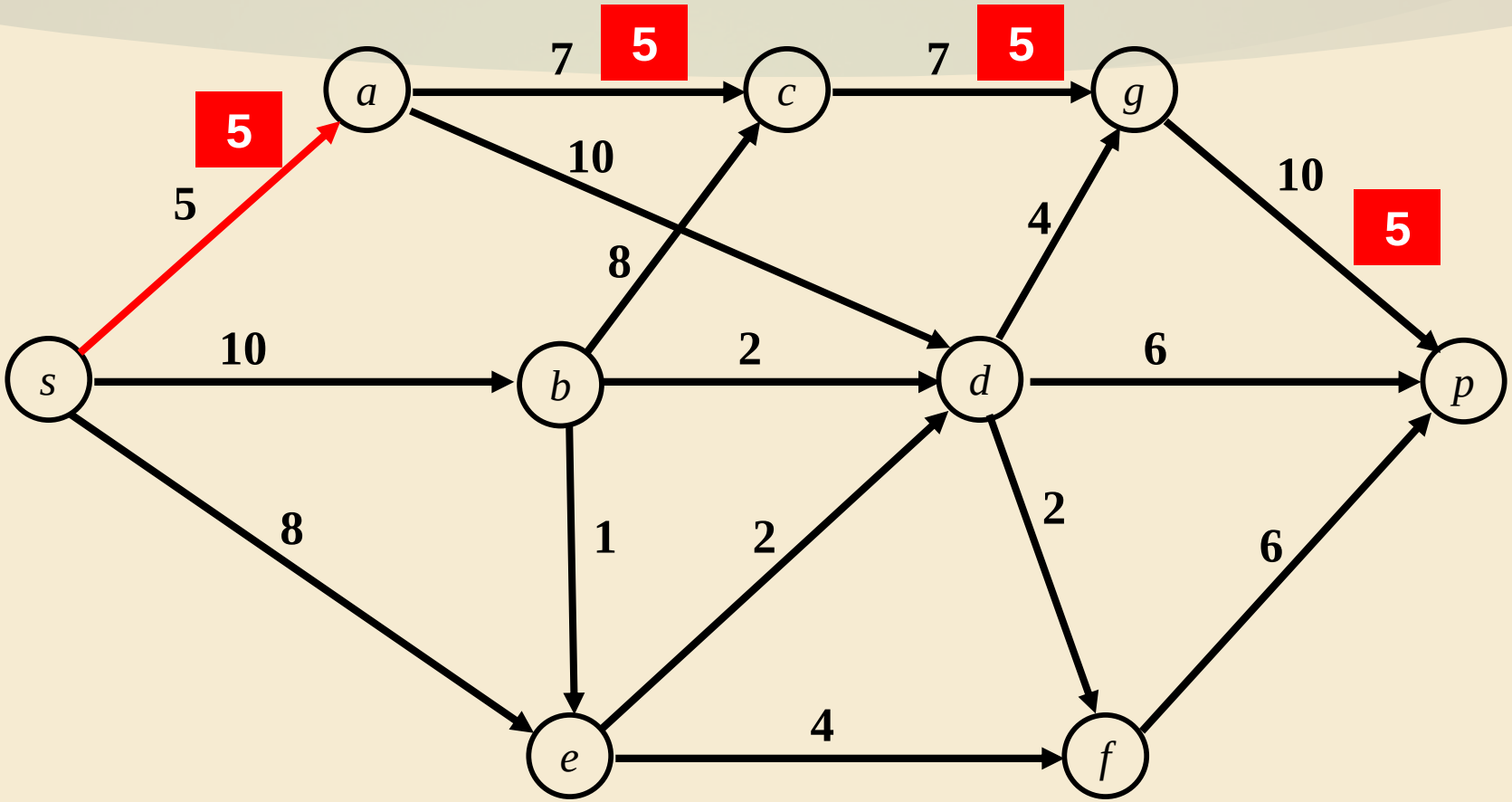
Exemple



- Chemin **(s,a,c,g,p) :** sa capacité résiduelle est **5**

Recherche d'un flot complet

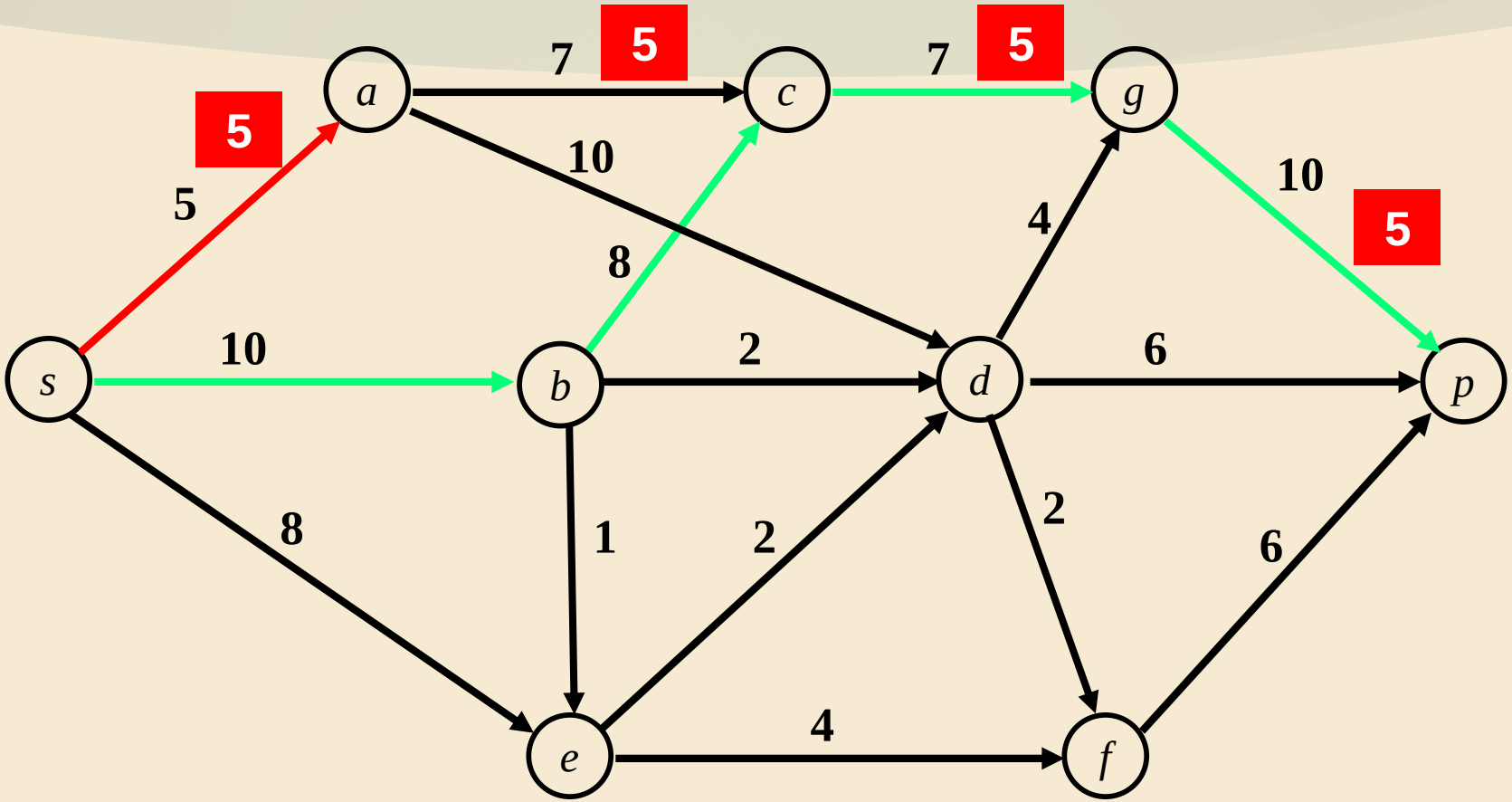
Exemple



- **Chemin (s,a,c,g,p) : on sature l'arc (s,a)**

Recherche d'un flot complet

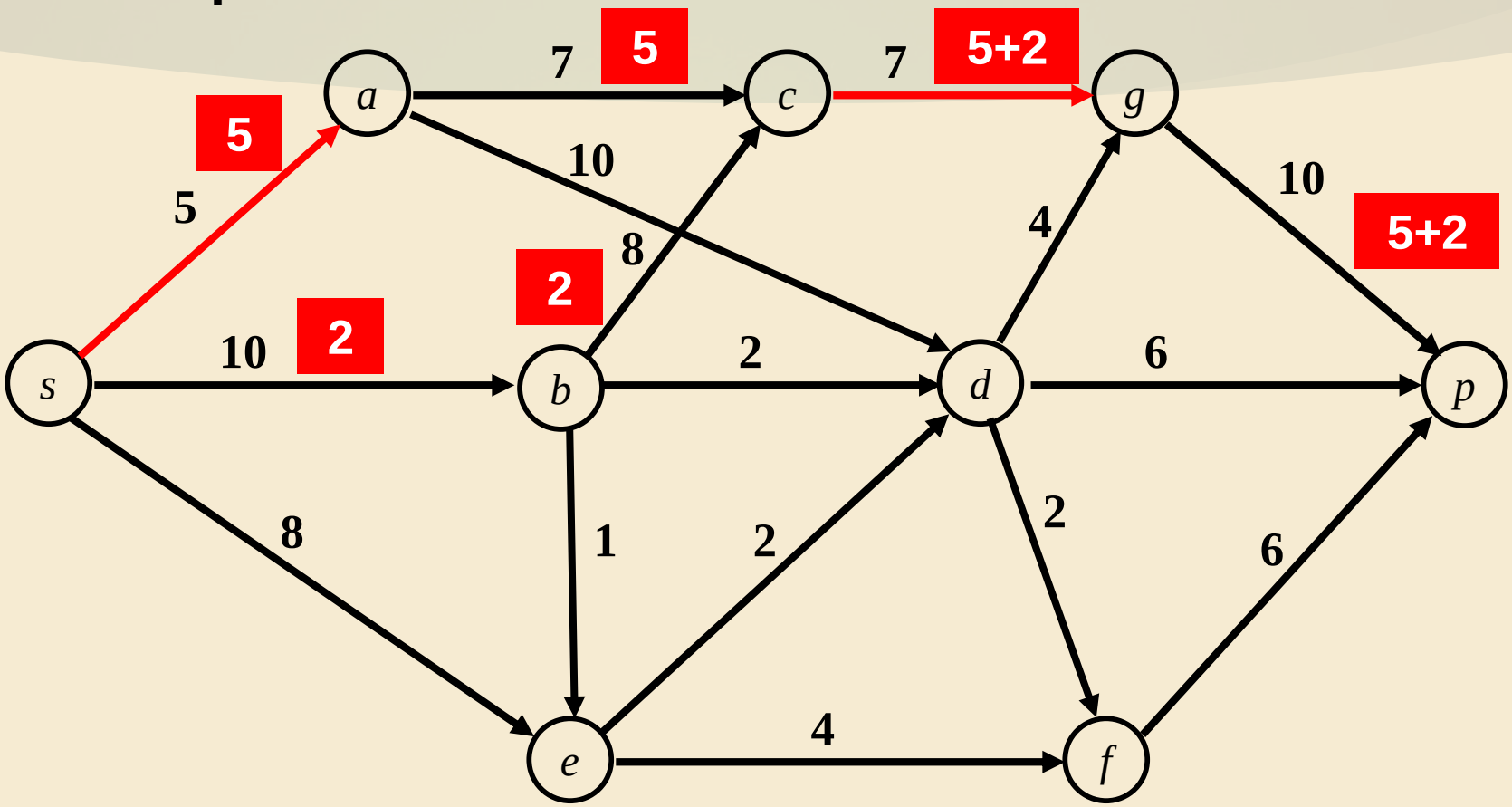
Exemple



- Chemin **(s,b,c,g,p) :** sa capacité résiduelle est **2 = 7-5**

Recherche d'un flot complet

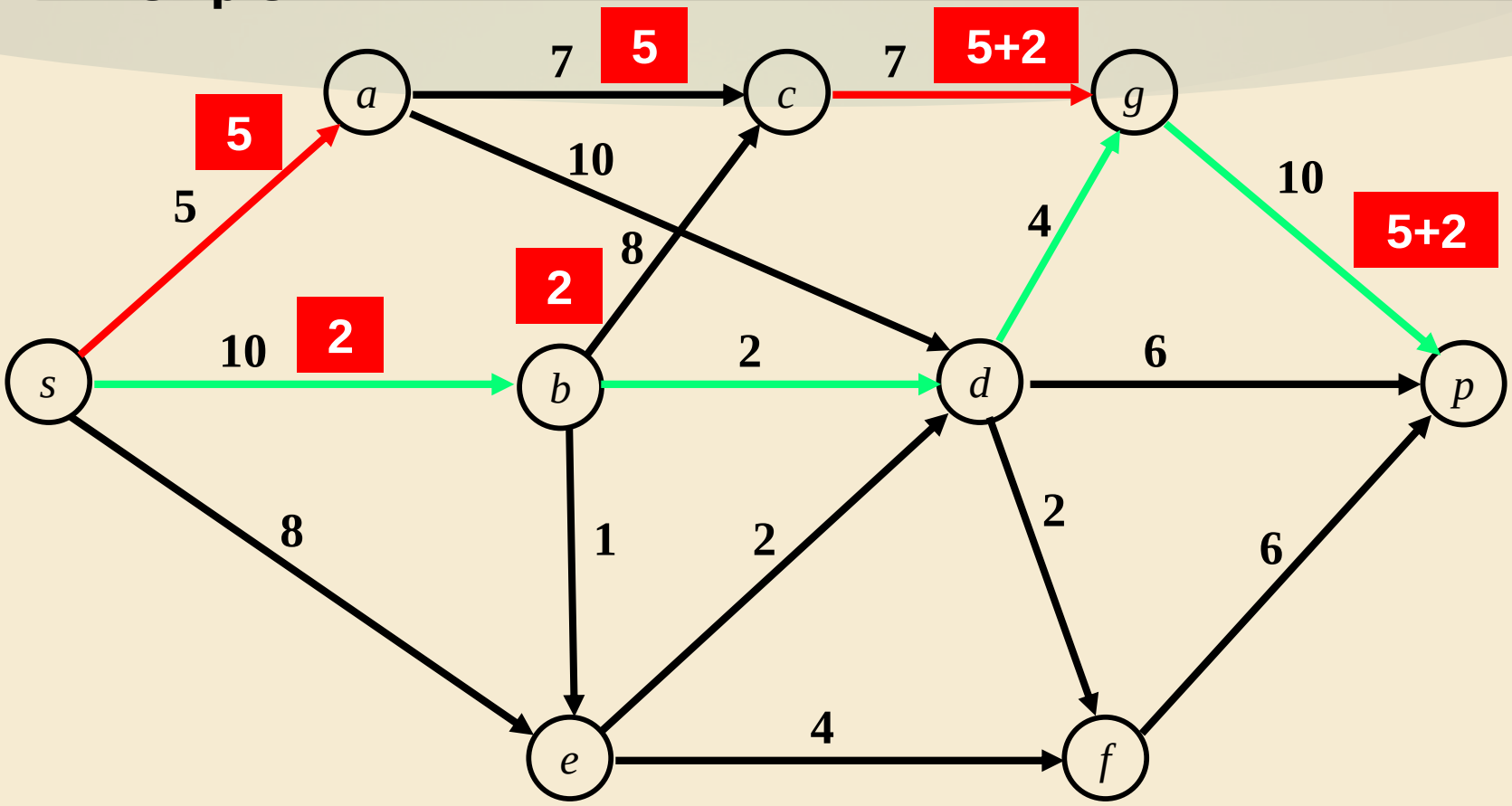
Exemple



- Chemin **(s,b,c,g,p)** : on sature l'arc **(c,g)**

Recherche d'un flot complet

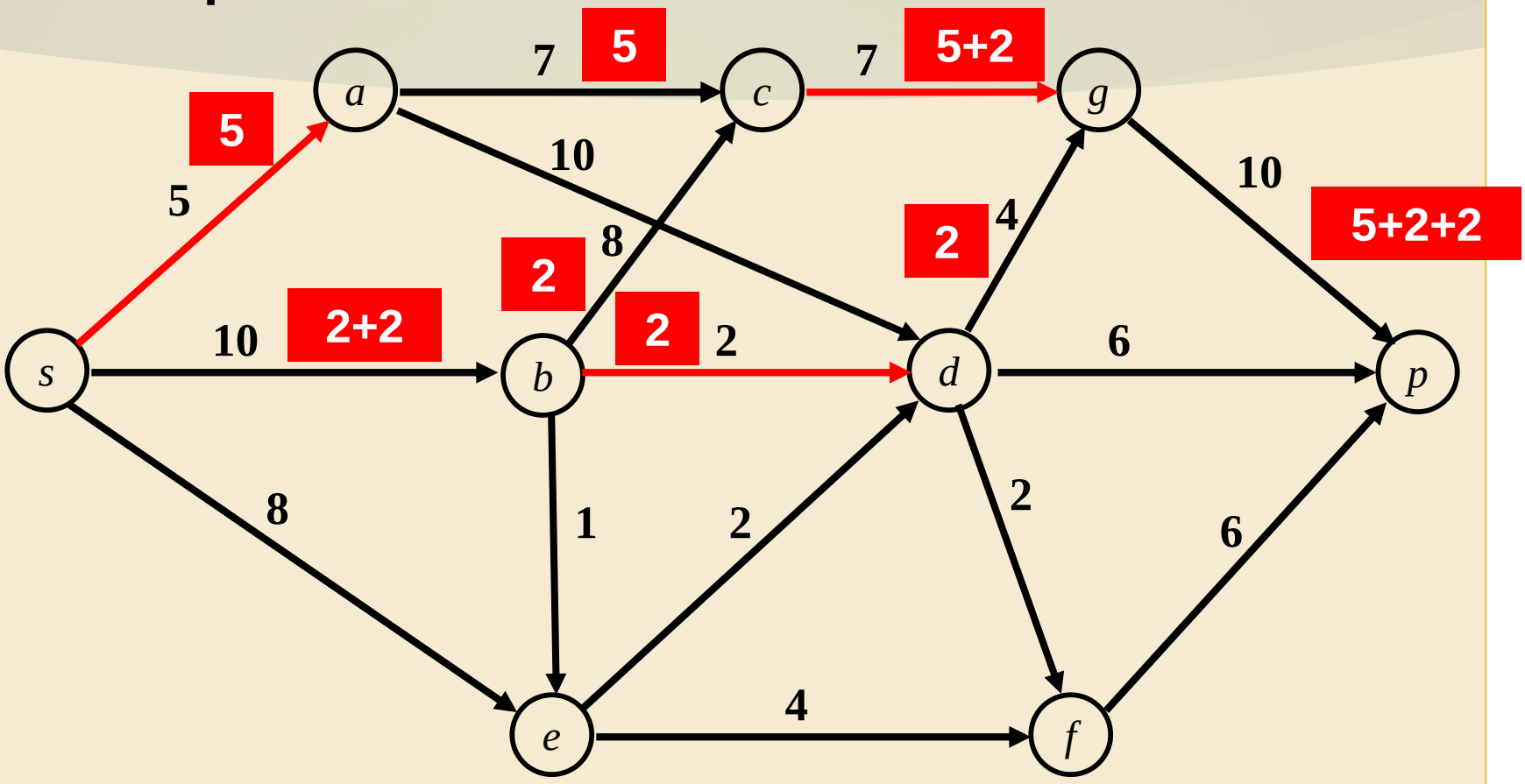
Exemple



- Chemin **(s,b,d,g,p)** : sa capacité résiduelle est **2**

Recherche d'un flot complet

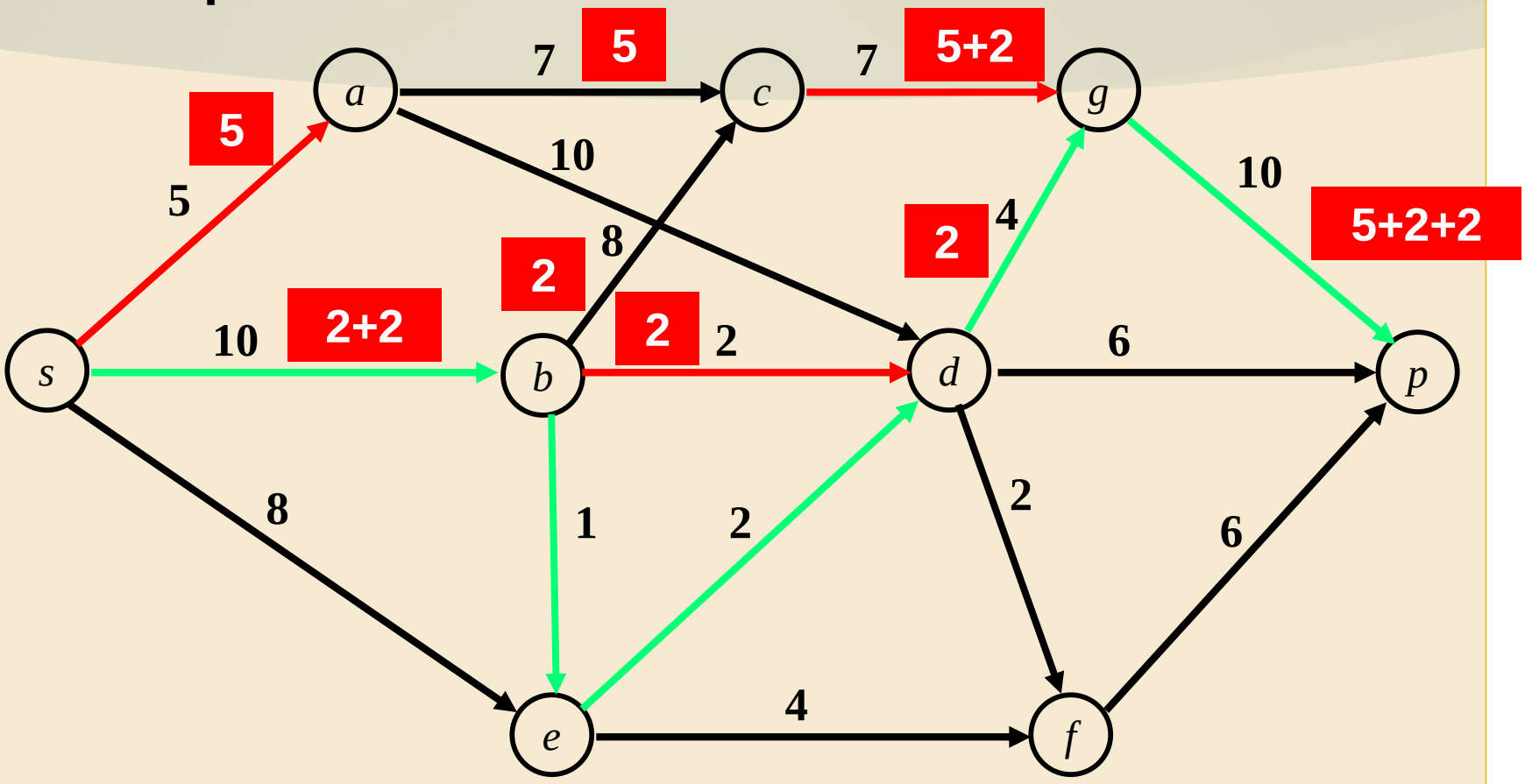
Exemple



- Chemin **(s,b,d,g,p)** : on sature l'arc **(b,d)**

Recherche d'un flot complet

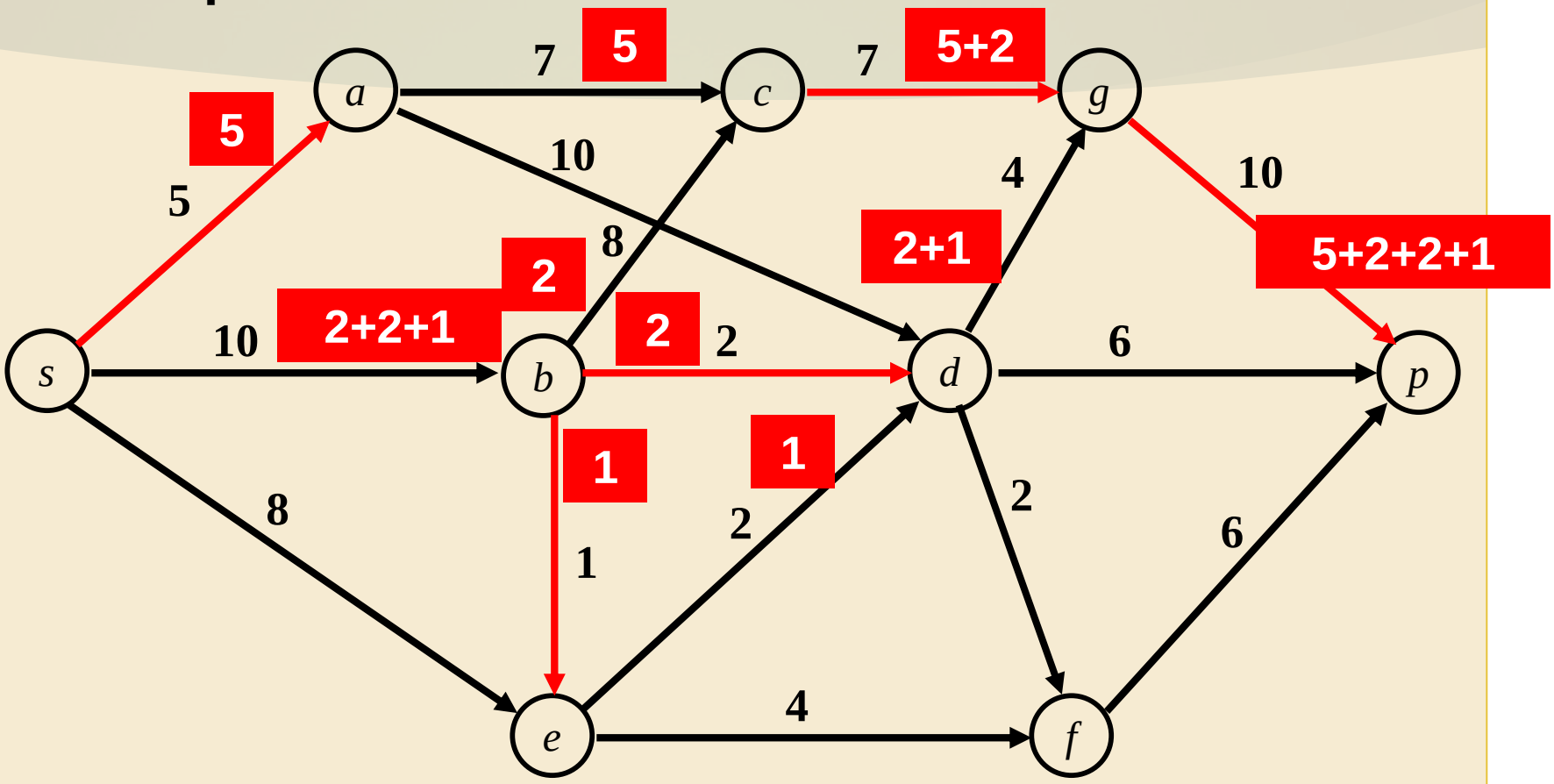
Exemple



- Chemin **(s,b,e,d,g,p)** : sa capacité résiduelle est **1**

Recherche d'un flot complet

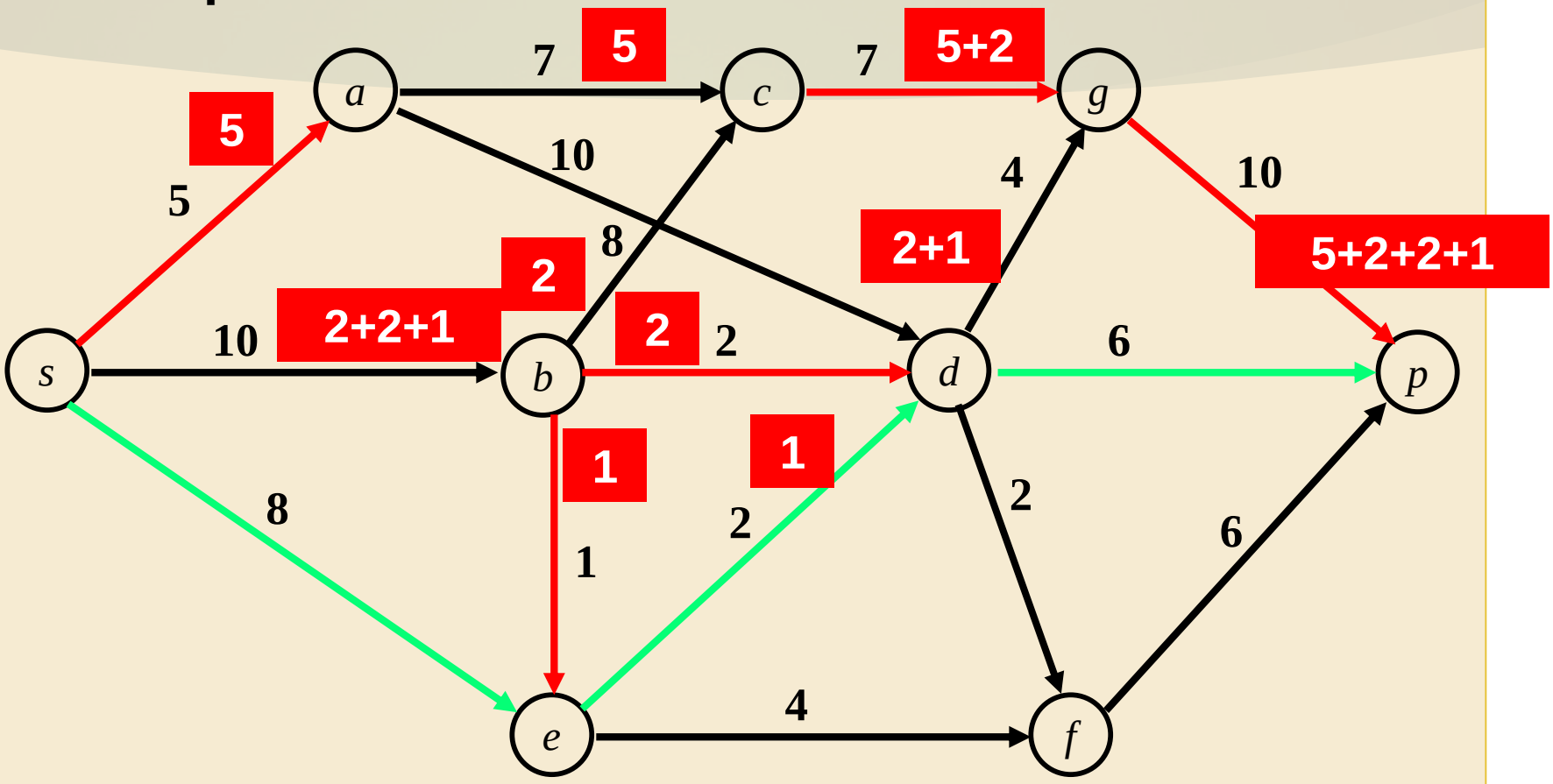
Exemple



- Chemin **(s,b,e,d,g,p)** : on sature l'arc **(b,e)** et l'arc **(g,p)**

Recherche d'un flot complet

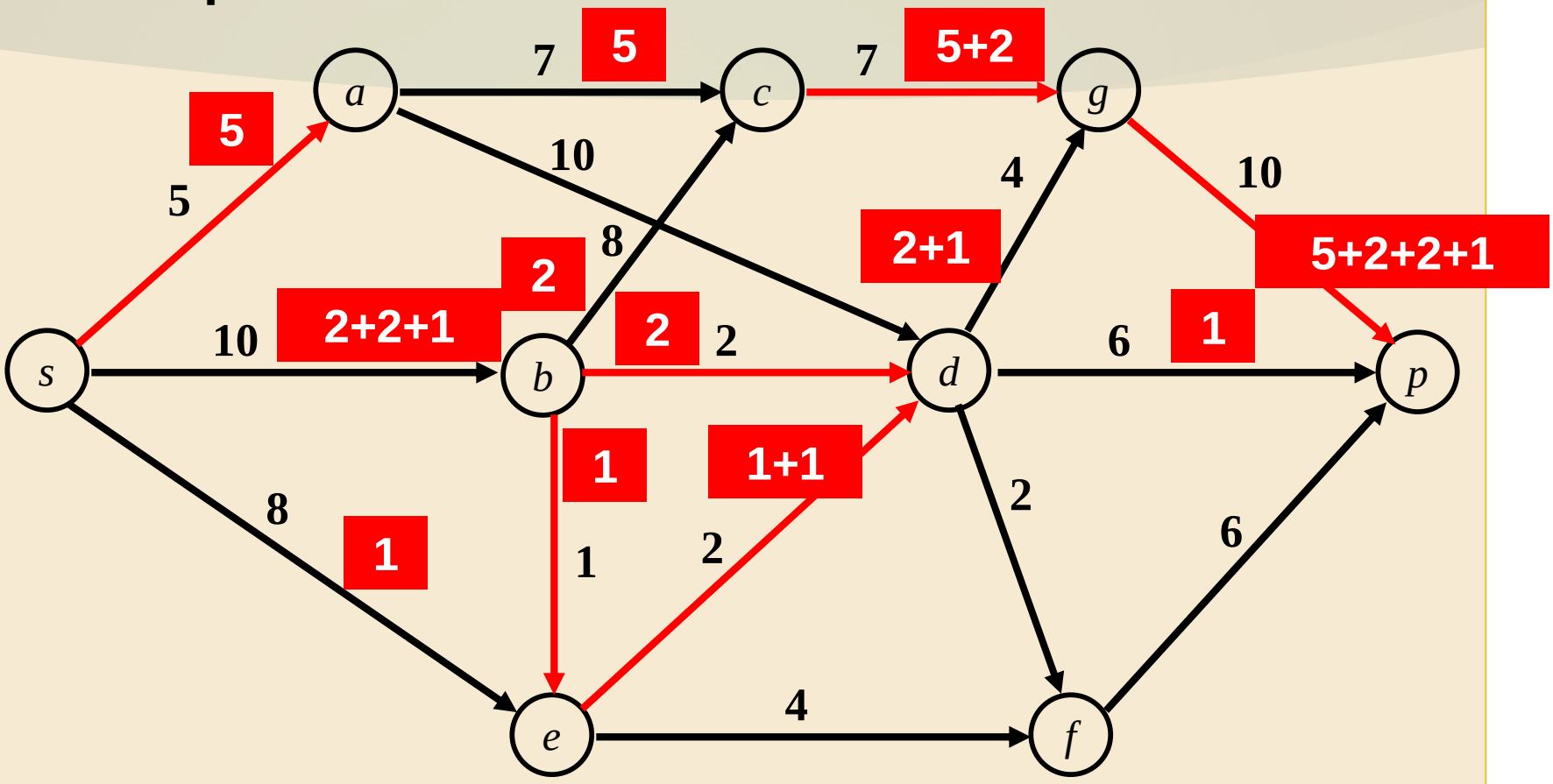
Exemple



- Chemin **(s,e,d,p)** : sa capacité résiduelle est **1**

Recherche d'un flot complet

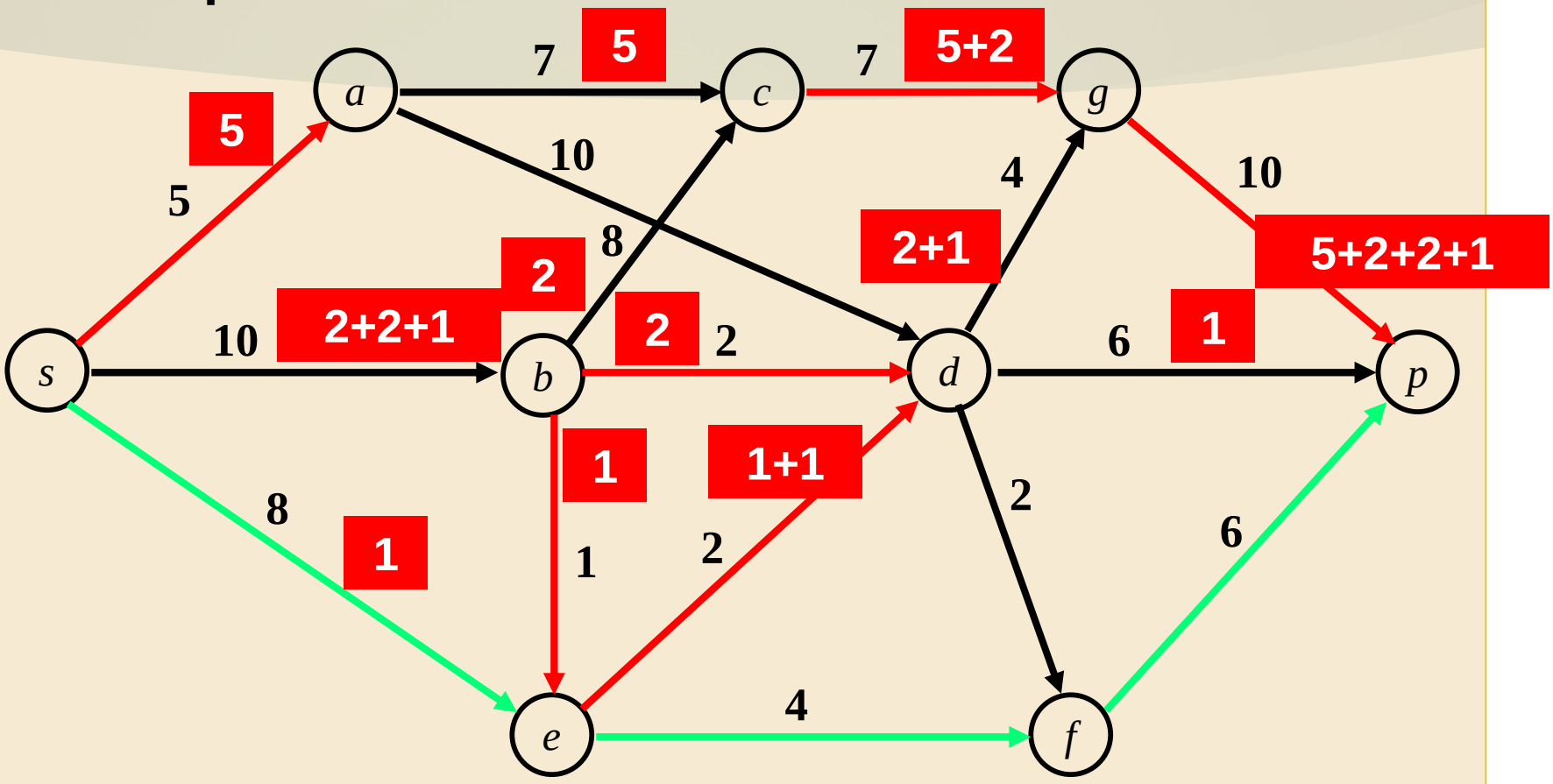
Exemple



- Chemin **(s,e,d,p)** : on sature l'arc **(e,d)**

Recherche d'un flot complet

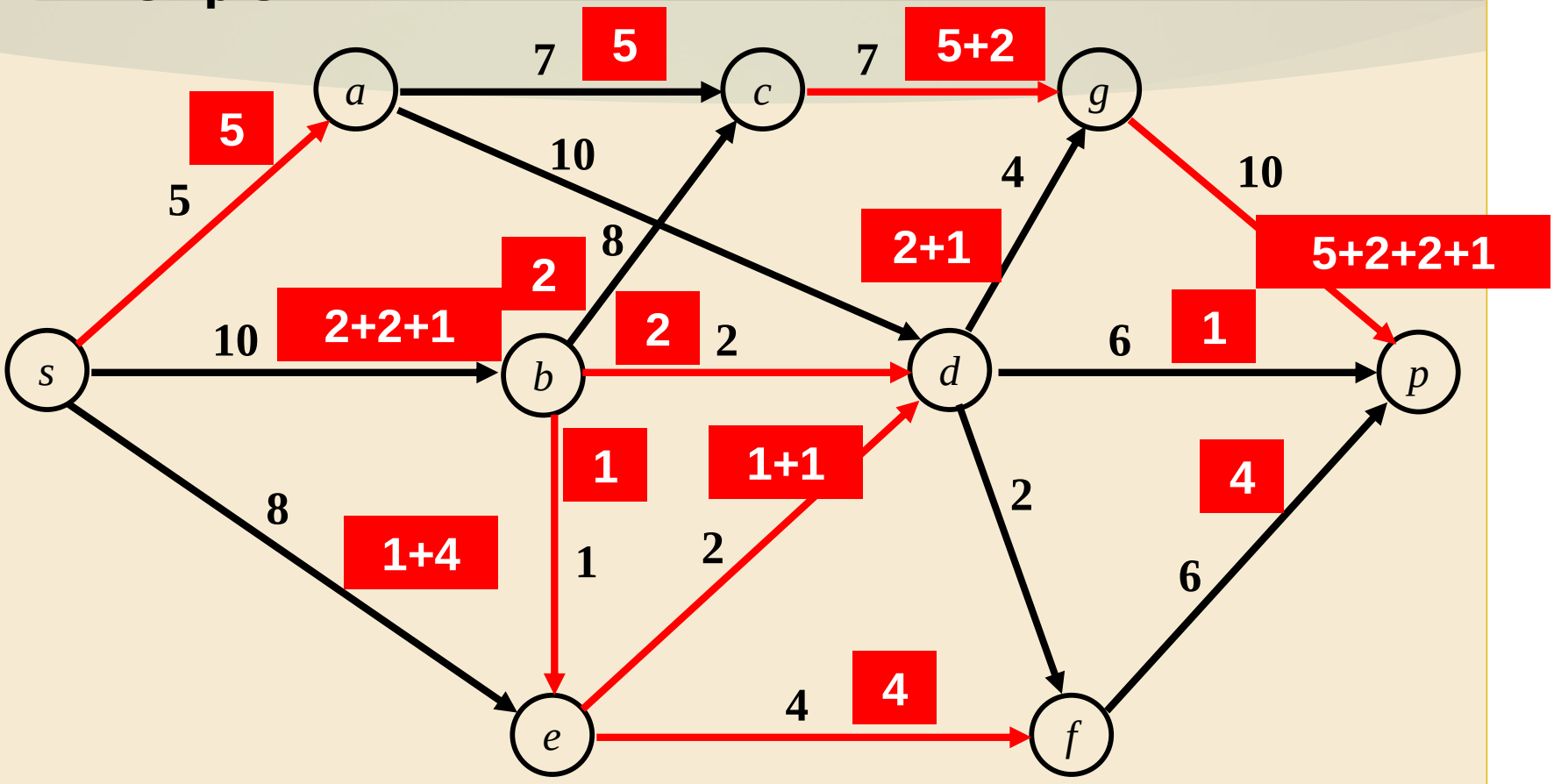
Exemple



- Chemin **(s,e,f,p)** : sa capacité résiduelle est **4**

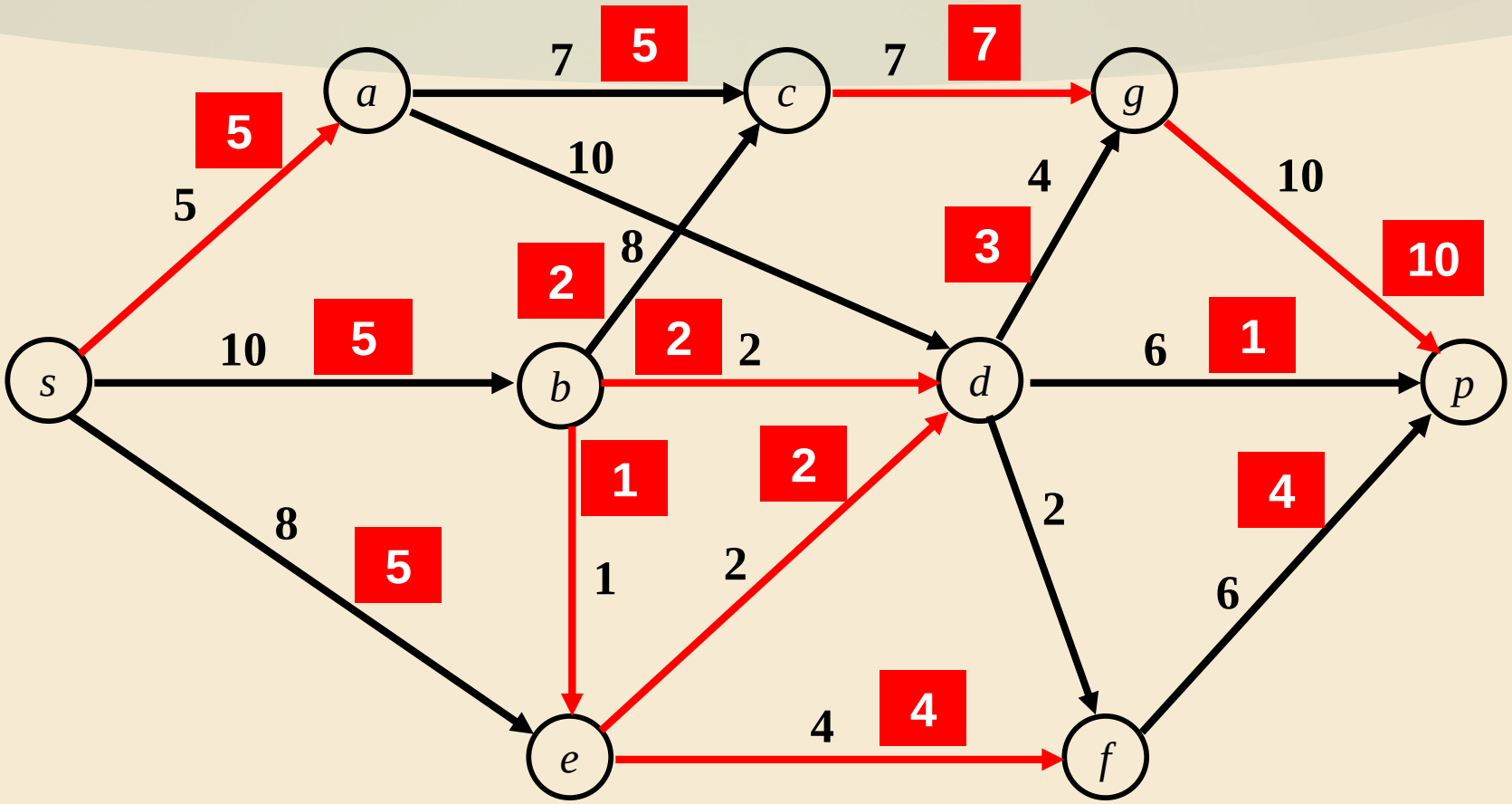
Recherche d'un flot complet

Exemple



- Chemin **(s,e,f,p)** : on sature l'arc **(e,f)**

Recherche d'un flot complet



- Tous les chemins sont saturés : on obtient **un flot complet**

Algorithme de Ford-Fulkerson

Définitions

- Une **chaîne améliorante** est une chaîne sur laquelle on peut augmenter le flux allant de **s** à **p**

- Plus précisément une chaîne améliorante est une chaîne élémentaire telle que :
  - Tout arc direct (x,y) de la chaîne n'est pas saturé
  - Tout arc indirect (x,y) de la chaîne a un flux positif

- Un arc (x,y) est direct si l'arc (x,y) appartient au réseau
- Un arc (x,y) est indirect si c'est l'arc (y,x) qui appartient au réseau

Algorithme de Ford-Fulkerson

Algorithme

- Initialisation :
  - l'entrée s est marquée «+»,
  - les autres sommets sont non marqués

- Tant que cela est possible choisir un sommet x vérifiant l'une des deux conditions suivantes:
  - x est l'extrémité d'un arc (y, x) non saturé tel que y est marqué **marquer «+» ce sommet**
  - x est l'origine d'un arc (x, y) tel que y est marqué et sur lequel circule un flux positif **marquer «-»ce sommet**

- indiquer le sommet qui a permis de marquer x, c'est-à-dire son «prédécesseur»: P(x)=y

Algorithme de Ford-Fulkerson

Algorithme

- si le puits **p** est **marqué**, la chaîne reliant les sommets marqués est **une chaîne améliorante**
- Si en fin d'application de cette procédure le puits **p n'est pas marqué,** il n'existe pas de chaîne améliorante et le **flot est optimal**.

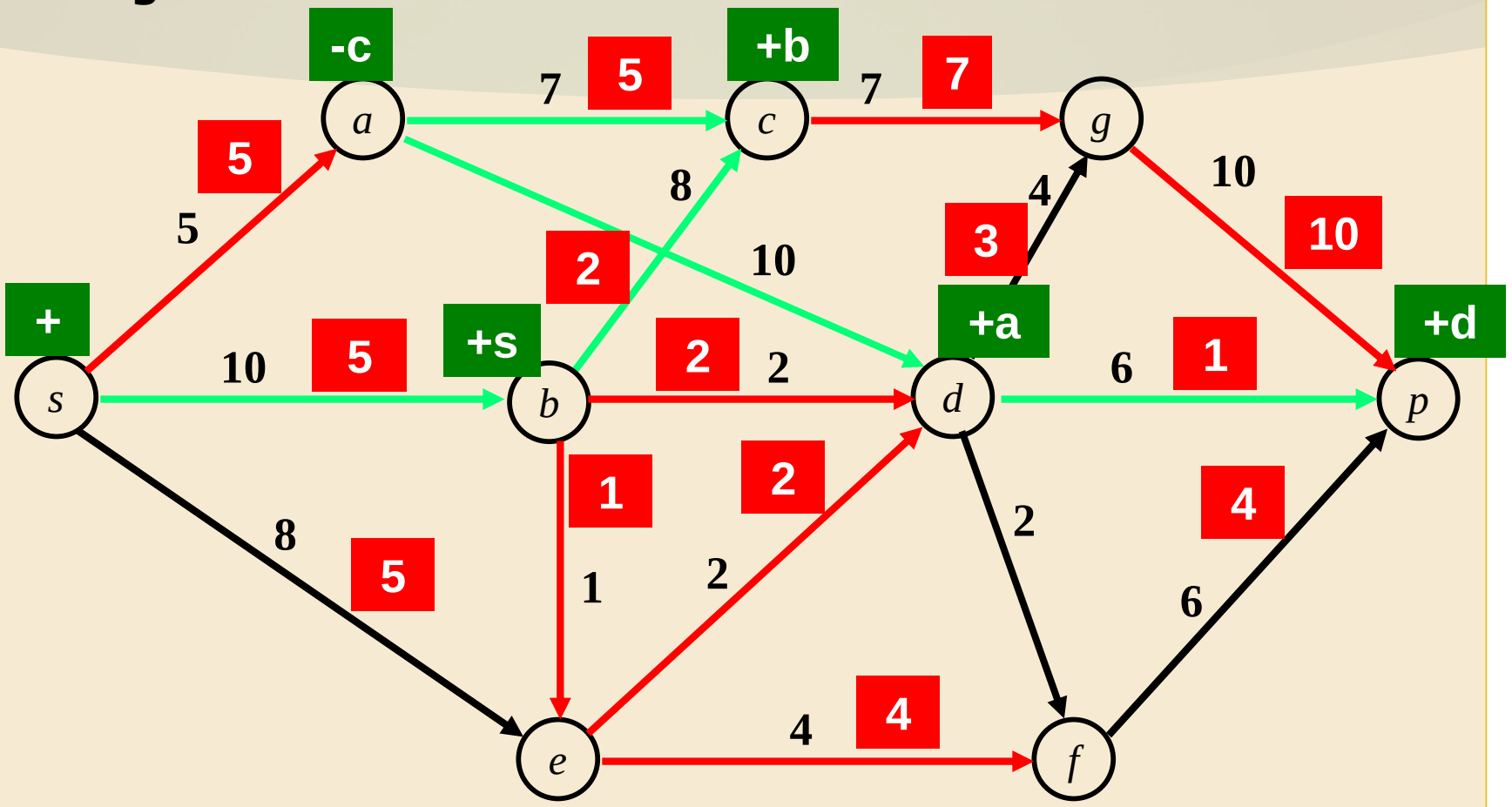
Recherche d'une chaîne améliorante

Algorithme

- 1ère itération :
  - On marque «+» le sommet s
  - L'arc (s,a) est saturé : on ne peut pas le marquer
  - L'arc (s,b) n'est pas saturé, s est marqué : on marque «+» le sommet b, on pose P(b)=s
  - L'arc (b,c) n'est pas saturé, b est marqué : on marque «+» le sommet c, on pose P(c)=b
  - L'arc (c,g) est saturé on ne peut le marquer, mais on peut retourner en arrière : c est marqué, sur l'arc (a,c) circule un flot positif, on marque «-» a, on pose P(a)=c
  - L'arc (a,d) n'est pas saturé, a est marqué ; on marque «+» le sommet d et on pose P(d)=a
  - L'arc (d,p) n'est pas saturé, d est marqué : on marque «+» le sommet p
- p est marqué : le flot n'est pas optimal

Recherche d'une chaîne améliorante

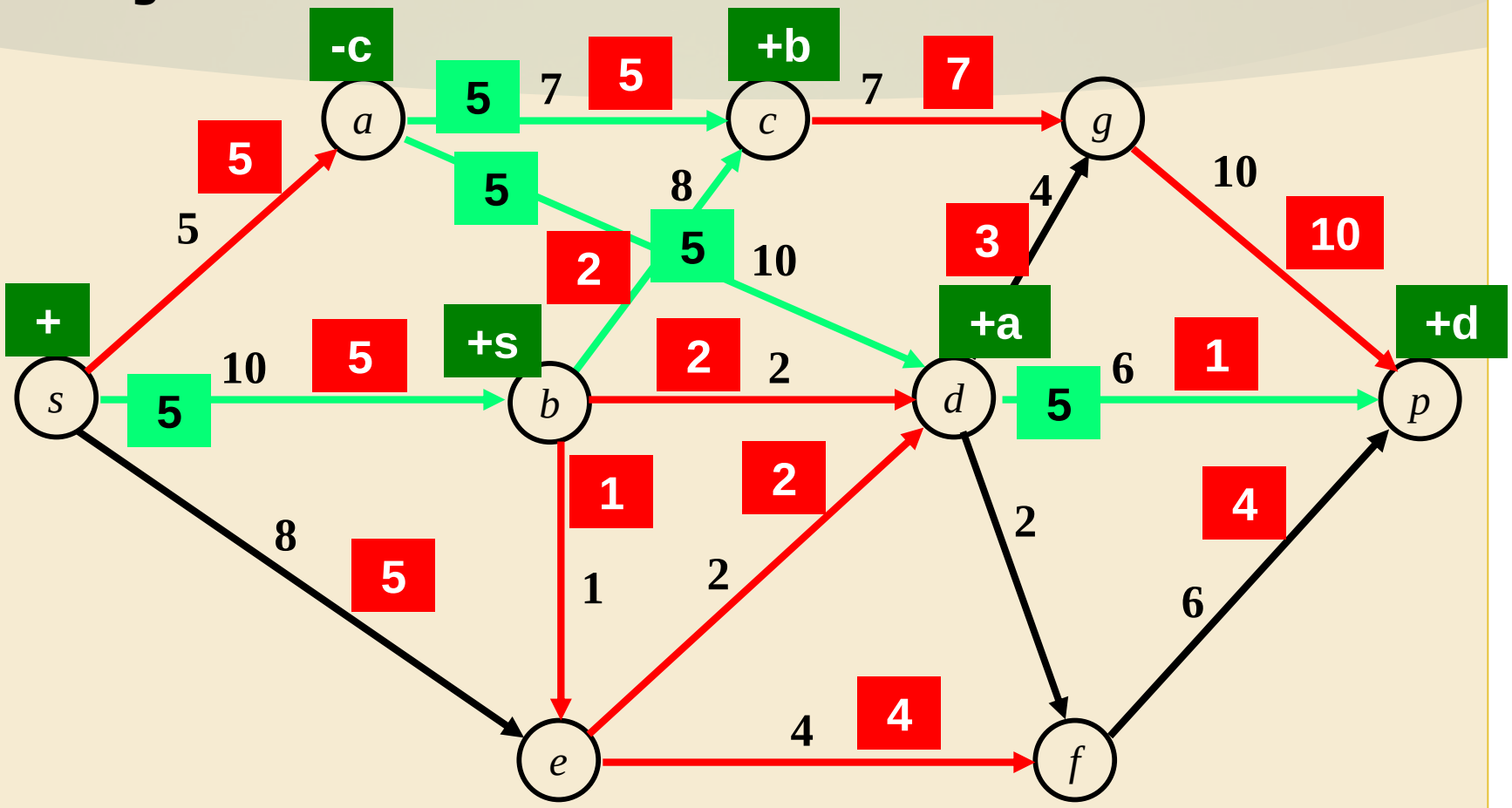
Algorithme



- Une chaîne améliorante : **(s,b,c,a,d,p)**

Recherche d'une chaîne améliorante

Algorithme



- Une chaîne améliorante : **(s,b,c,a,d,p)**

Recherche d'une chaîne améliorante

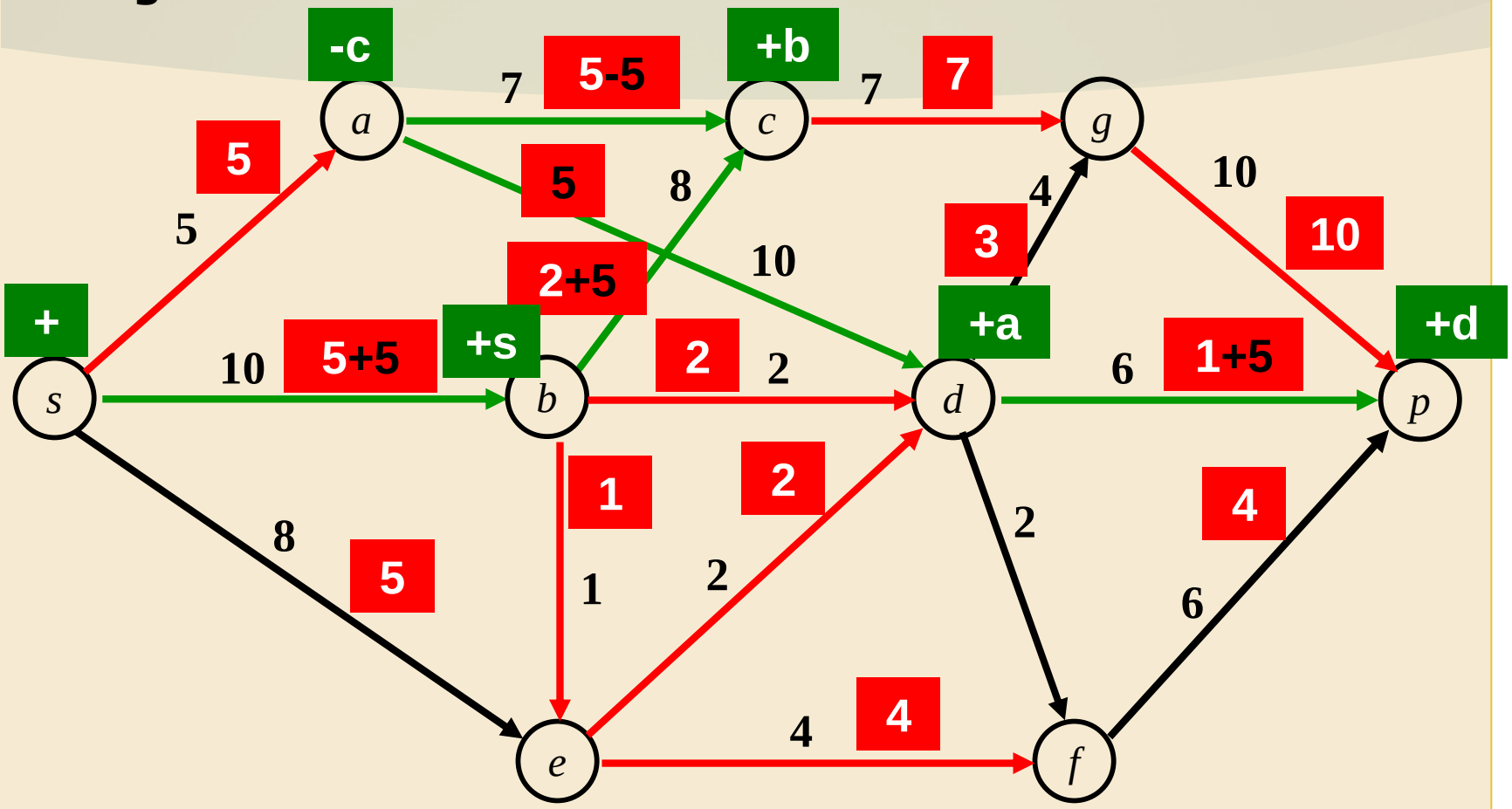
Algorithme

- La chaîne améliorante en sens inverse : (p, d, a, c, b, s)
- Les capacités résiduelles : 5, 6, 5, 10, 5
- On peut faire circuler 5 unités de flots supplémentaires
- On augmente le flot de l'arc (s, b) de 5 : l'arc est saturé
- On augmente le flot de l'arc (b, c) de 5
- Il arrive en c : 5+2+5 unités de flots alors que la capacité est de 7, il faut enlever 5 unités :
  - on les supprime sur l'arc (a,c)
  - on les fait passer par l'arc (a,d)

- On augmente ainsi le flot de l'arc (d,p) de 5 : l'arc est saturé

Recherche d'une chaîne améliorante

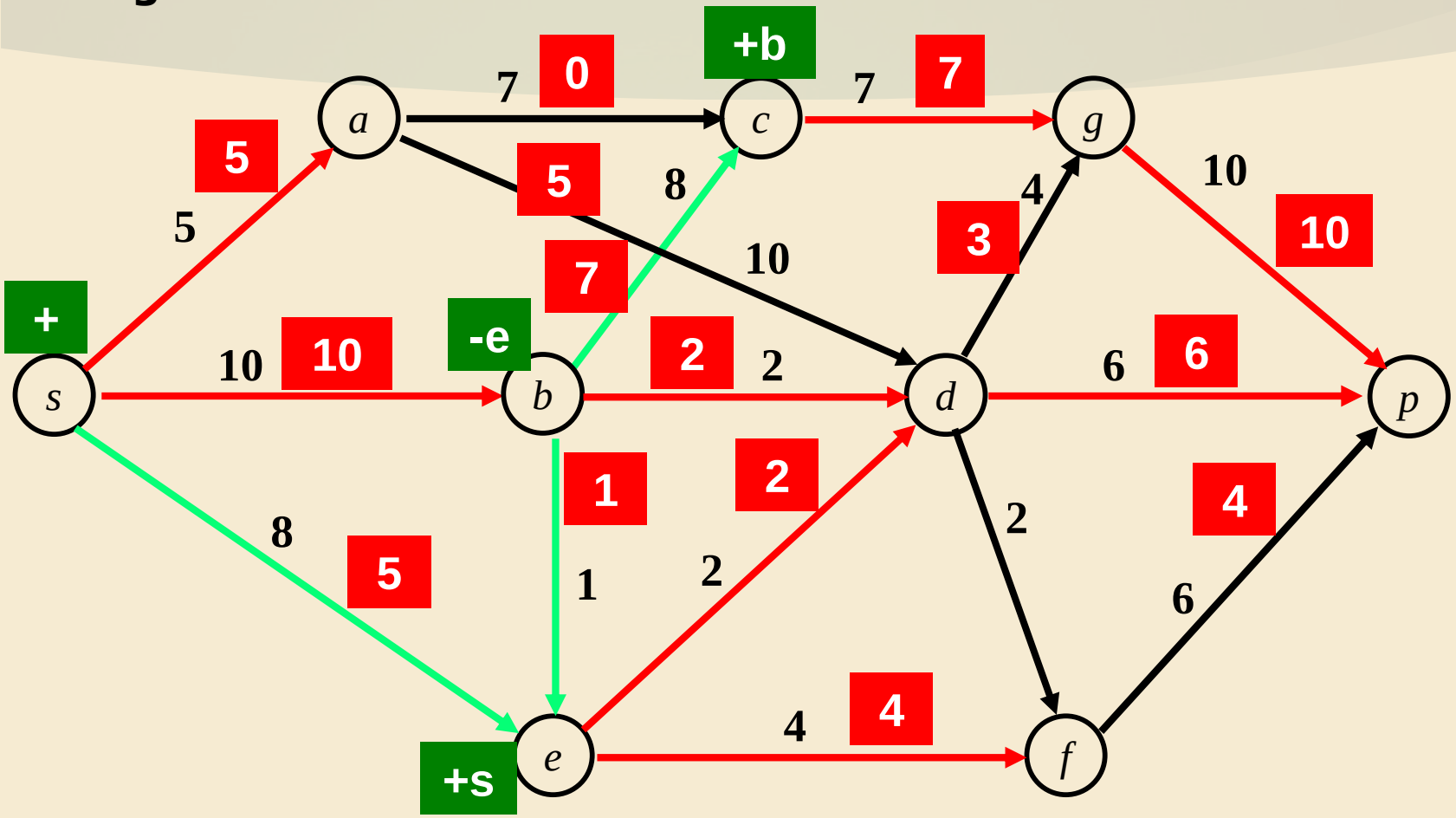
Algorithme



- Amélioration du flot initial

Recherche d'une chaîne améliorante

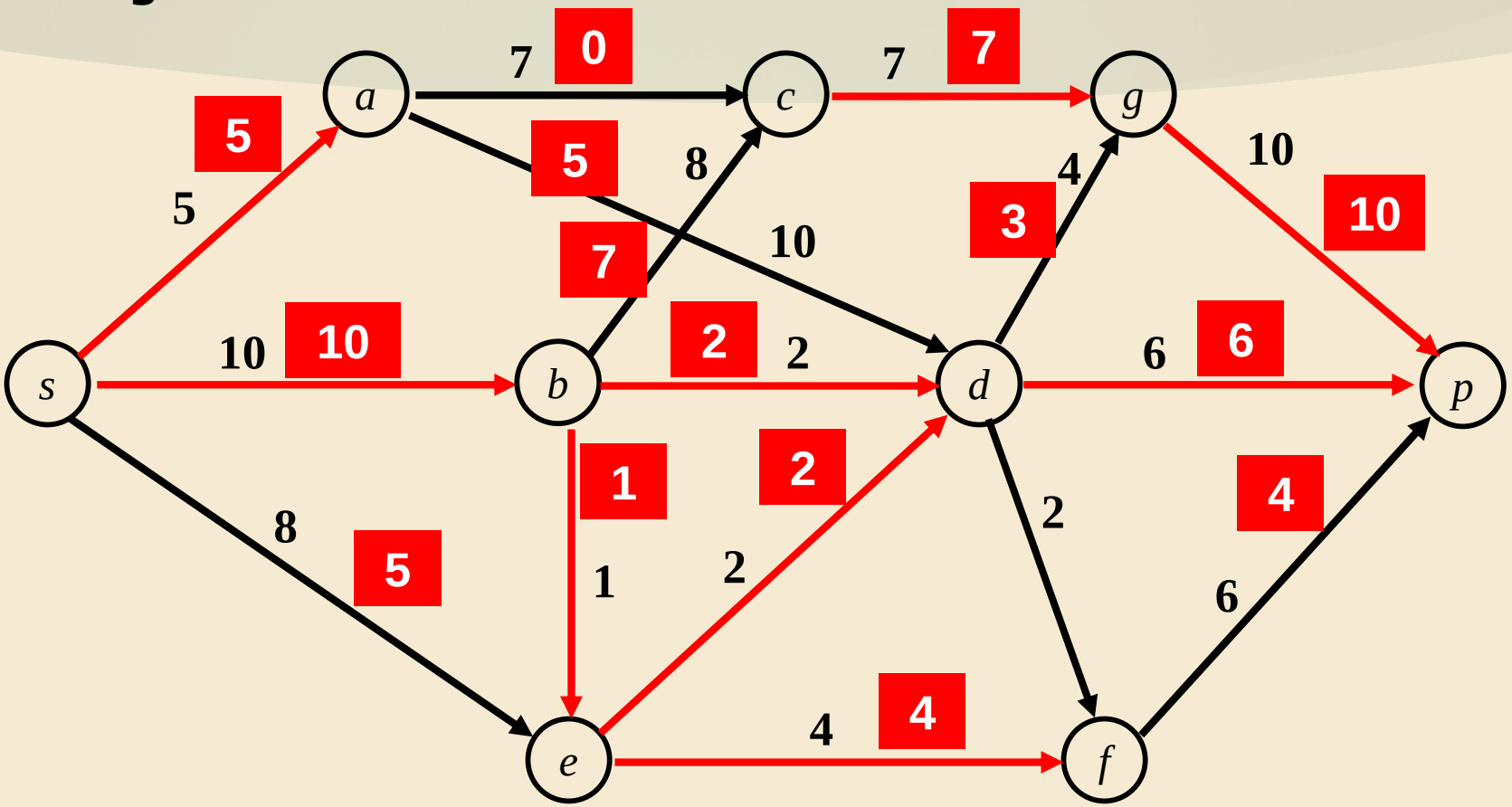
Algorithme



- p n'est pas marqué, l'optimum est atteint

Recherche d'une chaîne améliorante

Algorithme



- Un flot optimal

ORDONNANCEMENT SIMPLE

Définitions

Problème d'ordonnancement

- Un problème d'ordonnancement consiste à déterminer un calendrier de n tâches de manière à optimiser une certaine fonction objectif (exemple : durée totale du projet) tout en respectant certaines contraintes (antériorité de certaines tâches par rapport à d'autres, non simultanéité de certains travaux...)

Problème d'ordonnancement simple

- Un problème d'ordonnancement est dit simple si
  - les contraintes d'antériorité doivent toutes être satisfaites,
  - les durées des différentes tâches sont connues avec certitude,
  - les durées des différentes tâches sont indépendantes des dates de début et des moyens disponibles.

Exemple 1/2

- On souhaite définir un calendrier de tâches pour le problème suivant de manière à minimiser la durée totale du projet

Tâches	Descriptions	Durée (s)	Antériorité
1	Mettre la farine dans un saladier	**3**	**X**
2	Mettre deux œufs	**30**	**1 terminée**
3	Ajouter le lait et mélanger	**600**	**2 terminée**
4	Mettre de l'huile dans une poêle	**3**	**X**
5	Couper les bananes	**300**	**X**
6	Les mélanger à l'huile	**30**	**4 et 5 terminées**
8	Faire chauffer le mélange	**120**	**6 terminée**
8	Faire flamber	**10**	**7 terminée**
9	Faire cuire une crêpe	**10**	**3 terminée**
10	Verser le mélange sur la crêpe	**10**	**8 et 9 terminées**

- Comment modéliser ce problème?

Modélisation en graphe

Graphe potentiels-tâches

- Un problème d'ordonnancement simple peut être modélisé par un graphe potentiels-tâches (S,A) défini par :

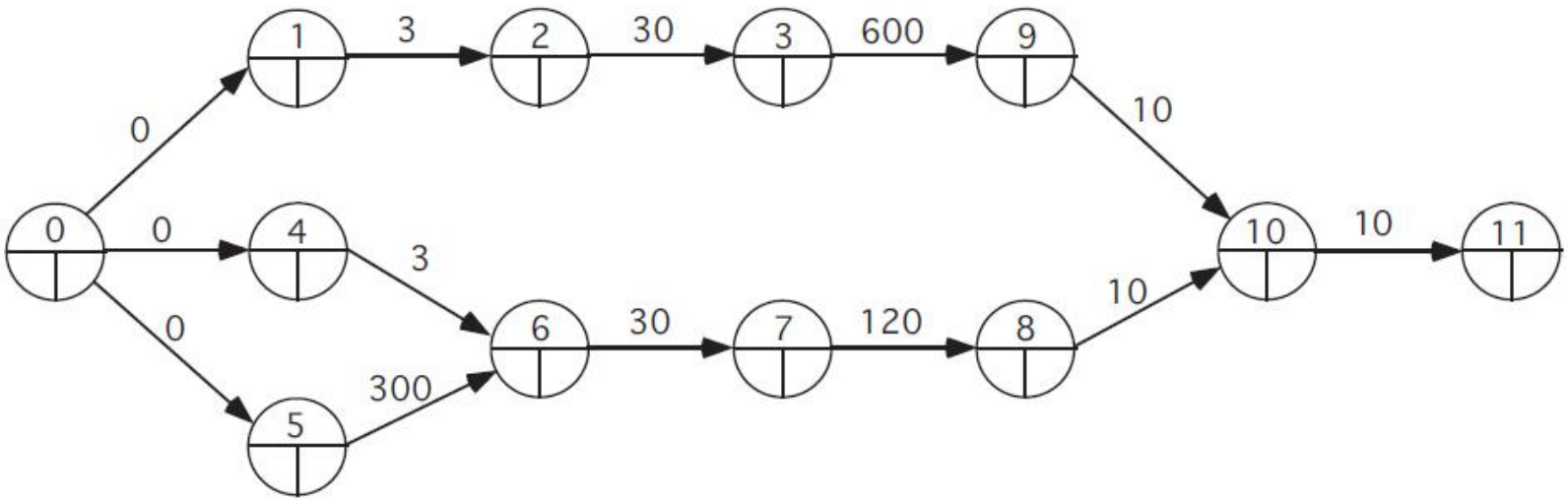
- S représente l'ensemble des tâches.
  - S = {1, . . . , n} [{0, n +1}, inclusion de deux tâches fictives 0 et n + 1 représentant le début et la fin du projet.

- A représente les contraintes d'antériorité.
  - Si une tâche j ne peut être réalisée avant la réalisation d'une tâche i , alors (i , j) $\in$ A
  - Un arc (i , j) est valué par les temps nécessaires entre le début de la tâche i et le début de la tâche j .

- Pour tout i $\in$ S \ {0}, (0, i) $\in$ A avec l(0, i)=0
- Pour tout i $\in$ S \ {n+1}, (i, n+1) $\in$ A avec $l(i,n+1)=d_i$ où d_i est la durée de la tâche i.
- On peut supprimer la plupart de ces arcs

Exemple modélisé

Exemple



- Les arcs (0, i) pour i ∈ {2,3,6,7,8,9,10,11} sont inutiles
- Les arcs (i,11) pour i ∈ {0,1,2,3,4,5,6,7,8,9} sont inutiles

Le calendrier au plus tôt

Définition

- Le calendrier au plus tôt indique les dates de début au plus tôt de chacune des tâches.

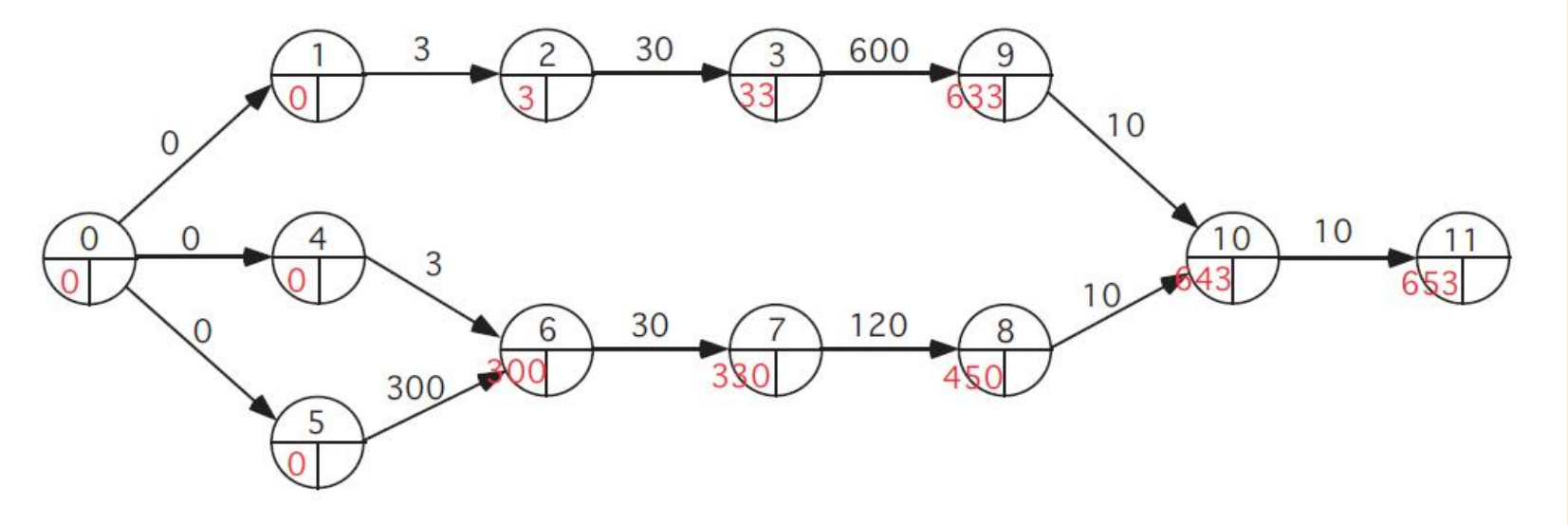
Calcul des dates au plus tôt

- On initialise le calendrier en fixant le début de la tâche 0 : $t^*_0 = 0$
- Pour les autres tâches j , deux cas possibles :
  - Il n'y a qu'un seul arc (i, j) $\in$ A, alors
    - $t^*_j = t^*_i + l(i, j)$
  - Il y a plusieurs arcs (i, j) $\in$ A, alors
    - $t^*_j = \max\{t^*_i + l(i, j) : (i, j) \in A\}$

- Le graphe est parcouru de "gauche à droite" pour ce calcul

Calendrier au plus tôt pour l'exemple

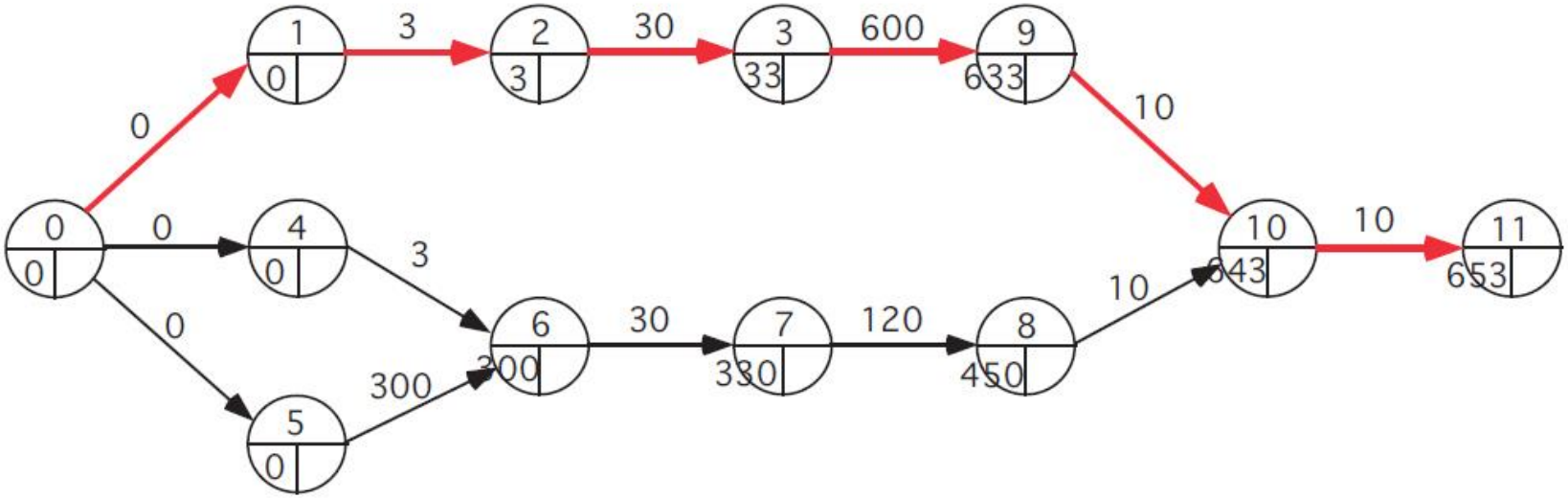
Exemple



- La date de début au plus tôt pour la tâche fictive de fin indique la durée minimale du projet

Calendrier au plus tôt pour l'exemple

Exemple



- Les arcs en rouge composent le plus long chemin du sommet de début au sommet de fin
- Ce chemin est appelé chemin critique, et les tâches le composant tâches critiques
- Ces tâches conditionnent la durée du projet, tout retard sur une tâche critique retarde la fin du projet

Calendrier au plus tard

Définition

- Etant donné une date de fin de projet fixée, un calendrier au plus tard indique les dates auxquels chaque tâche peut commencer au plus tard afin de respecter la date de fin.

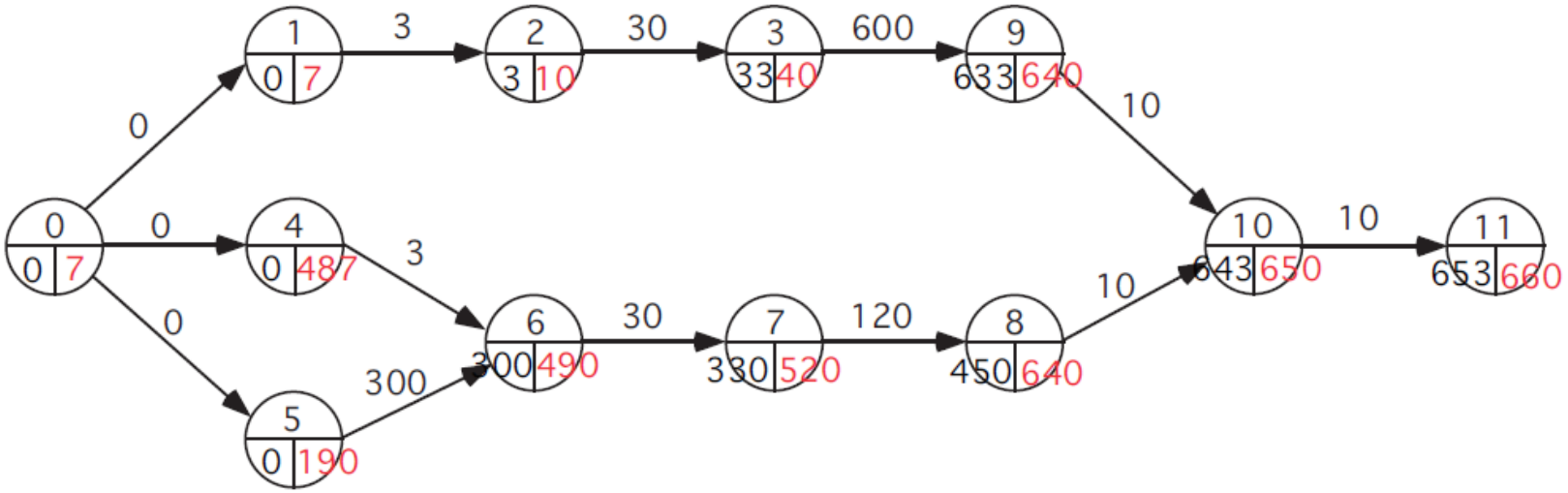
Calcul du calendrier au plus tard

- On initialise la date au plus tard de la tâche finale $\hat{t}_{n+1}$ avec la date de fin de projet fixée
- On a nécessairement $\hat{t}_{n+1} \geq t^*_{n+1}$ sinon le projet n'est pas réalisable
- Pour les autres tâches i , deux cas possibles :
  - Il n'y a qu'un seul arc $(i, j) \in A$, alors
    - $\hat{t}_i = \hat{t}_j - l(i, j)$
  - Il y a plusieurs arcs $(i, j) \in A$, alors
    - $\hat{t}_i = \min\{\hat{t}_j - l(i, j) : (i, j) \in A\}$
- Le graphe est parcouru de "droite à gauche" pour ce calcul

Calendrier au plus tard pour l'exemple

Exemple

- On fixe la date de fin de projet $t^{\wedge}_{11} = 660$



- Les dates au plus tard sont indiquées en rouge
- Afin de ne pas dépasser la date de fin de projet, toutes les tâches doivent débuter entre les dates de début au plus tôt et au plus tard

Marge

Définition

- Pour une tâche i , la différence $(\hat{t}_i - t^*_i)$ est appelée marge de la tâche i
- Cette marge est nulle pour les tâches critiques si
  - $\hat{t}_{n+1} = t^*_{n+1}$

Dans l'exemple

Tâche	Marge	Tâche	Marge
0	7	6	190
1	7	7	190
2	7	8	190
3	7	9	7
4	487	10	7
5	190	11	7

Diagramme de Gantt

Diagramme pour l'exemple

- On représente le projet sous la forme d'un diagramme de Gantt afin de mieux visualiser les tâches et leur durée, ainsi que leur exécution parallèle
- Dans ce diagramme, on a le temps en abscisse, et les ressources (représentant des tâches parallèles) en ordonnée

- Représentation du calendrier au plus tôt de l'exemple